

# NAVAL POSTGRADUATE SCHOOL

## Monterey, California



## THESIS

### INTEGRATION OF A SUBMARINE INTO NPSNET

by

Daniel Keith Bacon Jr.

September 1995

Thesis Advisor:

Michael J. Zyda

Thesis Co-Advisors:

Donald P. Brutzman

John S. Falby

**Approved for public release; distribution is unlimited.**



<b>REPORT DOCUMENTATION PAGE</b>			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time reviewing instructions, searching existing data sources gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
<b>1. AGENCY USE ONLY (Leave Blank)</b>		<b>2. REPORT DATE</b> September 1995	<b>3. REPORT TYPE AND DATES COVERED</b> Master's Thesis	
<b>4. TITLE AND SUBTITLE</b> INTEGRATION OF A SUBMARINE INTO NPSNET			<b>5. FUNDING NUMBERS</b>	
<b>6. AUTHOR(S)</b> Bacon, Daniel K., Jr.				
<b>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</b> Naval Postgraduate School Monterey, CA 93943-5000			<b>8. PERFORMING ORGANIZATION REPORT NUMBER</b>	
<b>9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)</b>			<b>10. SPONSORING/ MONITORING AGENCY REPORT NUMBER</b>	
<b>11. SUPPLEMENTARY NOTES</b> The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the United States Government.				
<b>12a. DISTRIBUTION / AVAILABILITY STATEMENT</b> Approved for public release; distribution is unlimited.			<b>12b. DISTRIBUTION CODE</b>	
<b>13. ABSTRACT</b> (Maximum 200 words) <p>In the current version of NPSNET there are two problems that prevent users of this virtual environment from achieving a realistic training experience. First, the motion of the vehicles is not built around realistic, physically-based models. In particular, the motion of computer-generated sea-going vehicles is not based on the hydrodynamic models that reflect the motion of actual ships moving through water. Second, vehicles in NPSNET are currently controlled by a single individual; they lack the capability to be controlled by a team. This misrepresents the many actual military vehicles—submarines, tanks, helicopters, and others—that must be controlled by <i>several</i> people working together.</p> <p>The approach taken was to update the submersible vehicle class in NPSNET in two ways. A physically-based hydrodynamic model was used to control the vehicle's motion through the virtual world. In addition, a network communications protocol was implemented to enable several remote individuals to control the same vehicle simultaneously.</p> <p>The result of this work is the creation of a computer-generated submersible vehicle whose motion is determined by a real-time hydrodynamic model so it moves through the virtual world according to physically-based models. This submersible is also capable of being controlled by several remote individuals—effectively the same team members who would perform the job in the actual vehicle. This ultimately results in a more realistic user experience as well as a more effective training tool for NPSNET.</p>				
<b>14. SUBJECT TERMS</b> Submarine Trainer Graphical Ocean Environment Mutiplayer Control			<b>15. NUMBER OF PAGES</b>	
			<b>16. PRICE CODE</b>	
<b>17. SECURITY CLASSIFICATION OF REPORT</b> Unclassified	<b>18. SECURITY CLASSIFICATION OF THIS PAGE</b> Unclassified	<b>19. SECURITY CLASSIFICATION OF ABSTRACT</b> Unclassified	<b>20. LIMITATION OF ABSTRACT</b> UL	



**Approved for public release; distribution is unlimited**

## **INTEGRATION OF A SUBMARINE INTO NPSNET**

Daniel Keith Bacon Jr.  
Liutenant, United States Navy  
B.S., United States Naval Academy, 1988

Submitted in partial fulfillment of the  
requirements for the degree of

**MASTER OF SCIENCE IN COMPUTER SCIENCE**

from the

**NAVAL POSTGRADUATE SCHOOL**

**September 1995**

Author:

---

Daniel Keith Bacon Jr.

Approved by:

---

Michael J. Zyda, Thesis Advisor

---

Donald P. Brutzman, Thesis Co-Advisor

---

John S. Falby, Thesis Co-Advisor

---

Ted Lewis, Chairman,  
Department of Computer Science



## ABSTRACT

In the current version of NPSNET there are two problems that prevent users of this virtual environment from achieving a realistic training experience. First, the motion of the vehicles is not built around realistic, physically-based models. In particular, the motion of computer-generated sea-going vehicles is not based on the hydrodynamic models that reflect the motion of actual ships moving through water. Second, vehicles in NPSNET are currently controlled by a single individual; they lack the capability to be controlled by a team. This misrepresents the many actual military vehicles—submarines, tanks, helicopters, and others—that must be controlled by *several* people working together.

The approach taken was to update the submersible vehicle class in NPSNET in two ways. A physically-based hydrodynamic model was used to control the vehicle's motion through the virtual world. In addition, a network communications protocol was implemented to enable several remote individuals to control the same vehicle simultaneously.

The result of this work is the creation of a computer-generated submersible vehicle whose motion is determined by a real-time hydrodynamic model so it moves through the virtual world according to physically-based models. This submersible is also capable of being controlled by several remote individuals—effectively the same team members who would perform the job in the actual vehicle. This ultimately results in a more realistic user experience as well as a more effective training tool for NPSNET.





# TABLE OF CONTENTS

I.	INTRODUCTION .....	1
	A. BACKGROUND .....	1
	B. MOTIVATION .....	2
	1. Littoral Warfare and the Submarine .....	2
	2. Teamwork vs. Individual Training .....	3
	C. OBJECTIVES .....	4
	D. THESIS OUTLINE.....	5
II.	RELATED WORK .....	7
	A. NPSNET .....	7
	B. HYDRODYNAMICS .....	8
	C. MULTI-PLAYER CONTROL .....	10
	D. GRAPHICAL REPRESENTATION OF AN OCEAN ENVIRONMENT....	11
III.	HYDRODYNAMIC MODEL .....	13
	A. INTRODUCTION .....	13
	B. BUOYANCY FUNCTION.....	14
	1. Buoyancy in General.....	14
	2. Function Based on Depth.....	15
	3. Buoyancy Function Based on Ship's Angle .....	16
	C. IMPLEMENTATION.....	18
	1. New Coefficients .....	18
	2. Coordinate System .....	19
	a. NPSNET .....	20
	b. Hydrodynamics Model.....	20
	c. Integration of two reference models .....	21
	3. Inputs and Outputs .....	22
IV.	MULTI-CONTROLLER PROTOCOL .....	23

A. INTRODUCTION .....	23
B. SUBMARINE CONTROL PANELS .....	24
1. Helm Control Panel.....	24
2. Officer of the Deck (OOD) Control Panel.....	24
3. Weapons Officer Control Panel.....	26
C. COMMUNICATION PROTOCOL.....	28
V. GRAPHICAL REPRESENTATION OF OCEAN ENVIRONMENT.....	31
A. INTRODUCTION .....	31
B. GRAPHICAL UNDERSEA ENVIRONMENT .....	32
C. OCEAN WAVE CARPET .....	34
VI. CONCLUSION AND RECOMMENDATIONS .....	39
A. RUNTIME PERFORMANCE.....	39
1. HYDRODYNAMICS MODEL.....	39
2. OCEAN WAVE CARPET .....	39
B. CONCLUSIONS.....	40
C. LIMITATIONS.....	40
D. RECOMMENDATIONS FOR FUTURE WORK .....	40
DARPA SUBOFF MODEL COEFFICIENTS .....	43
GENERIC SUBMARINE MODEL COEFFICIENTS.....	59
OBTAINING NPSNET SOURCE CODE.....	75
LIST OF REFERENCES .....	77
INITIAL DISTRIBUTION LIST .....	79

## LIST OF FIGURES

Figure 1. Virtual Anti Submarine Warfare Exercise .....	3
Figure 2. NPS Autonomous Underwater Vehicle (AUV) .....	9
Figure 3. Buoyancy related dimensions.....	15
Figure 4. Buoyancy versus keel depth for a rectangular box (solid line) and a cylinder or ellipsoid (dotted line) .....	16
Figure 5. Moment arm created by change in CB position. ....	17
Figure 6. Surface Length determination. ....	18
Figure 7. NPSNET and Hydrodynamics Model coordinate conventions .....	21
Figure 8. Helm Control Panel .....	25
Figure 9. OOD Control Panel .....	27
Figure 10. Weapons Officer Control Panel.....	29
Figure 11. Undersea Environment .....	33
Figure 12. Ocean Carpet (Wire frame) .....	34
Figure 13. Ocean Carpet (Textured) .....	34
Figure 14. Time Lapsed View from Periscope with Ocean Wave Carpet.....	36
Figure 15. Time Lapsed View from Periscope with Ocean Carpet .....	37
Figure 16. Time Lapsed View from Periscope with Ocean Wave Carpet.....	38



# **I. INTRODUCTION**

## **A. BACKGROUND**

In 1990, researchers and students began work on what was to become the Naval Postgraduate School Networked Virtual Environment (NPSNET) at the Graphics and Video Laboratory of the Department of Computer Science at the Naval Postgraduate School in Monterey, California. NPSNET is an interactive distributed virtual simulation of military maneuvers. NPSNET's functionality and capabilities have improved with each generation of software, networking technology and graphic capabilities. [Zyda94]

Currently, NPSNET is in its fourth major version (NPSNET-IV). It includes a suite of complementary software applications such as network management tools and various interfaces. NPSNET uses the Distributed Interactive Systems (DIS) Protocol version 2.0.3 for networked communications[Zeswitz93]

The software applications of NPSNET graphically simulate many actual military activities, such as helicopter and tank warfare, fighter plane missions and foot soldier operations. Each of these electronic simulators--whether a ship, a bomber, or soldier--can run on separate workstations manufactured by Silicon Graphics Incorporated (SGI). Individuals--ideally trainees--can operate these software applications without difficulty.

One of the elements that makes NPSNET unique is the fact that the applications are networked and offer real-time interactions among the software applications. The DIS protocol allows for real-time three-dimensional (3-D) contacts and exchanges among all participants located at individual, geographically diverse workstations.

This thesis is an effort to add to the ongoing work that continually improves NPSNET. Specifically, a submarine entity with physically-based hydrodynamic motion and controls that can be run by three distant users simultaneously has been created.

## **B. MOTIVATION**

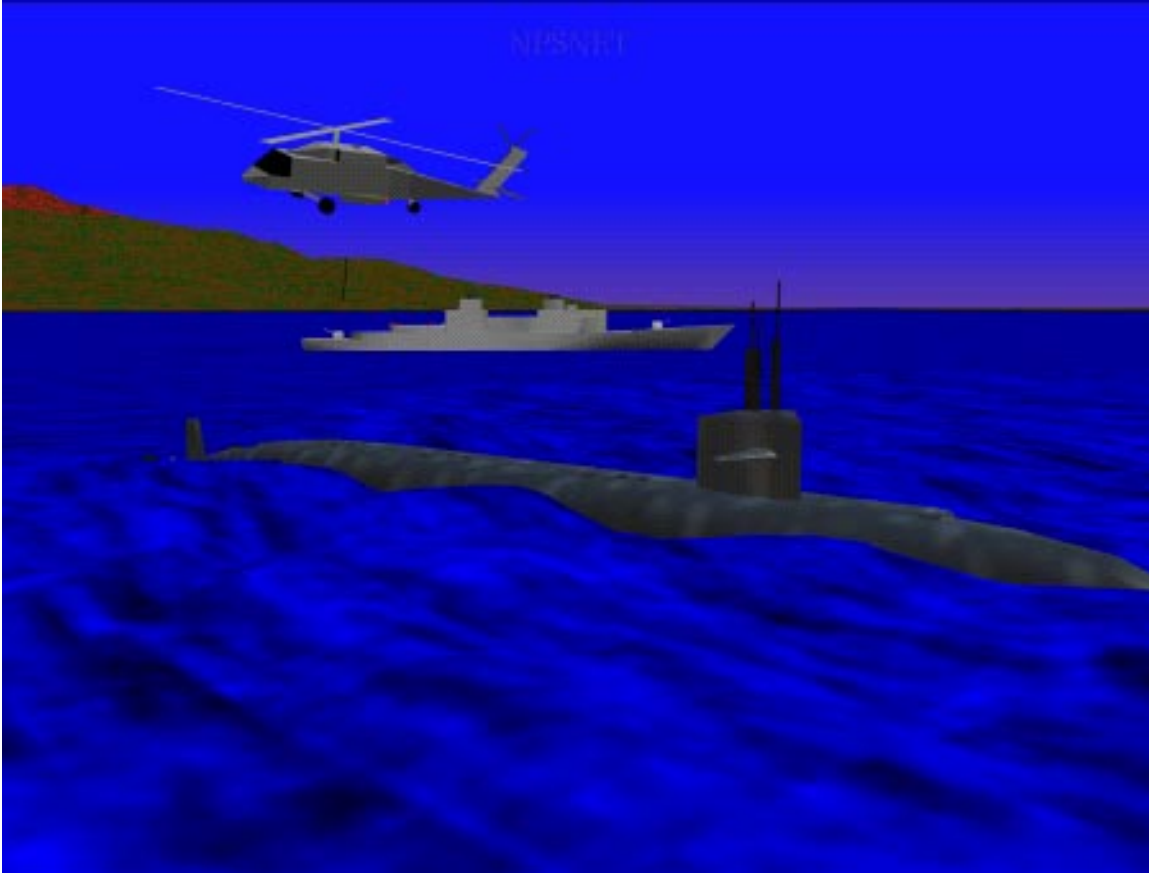
Computer simulation provides the Navy with an inexpensive yet highly realistic method of training. For proper training, various entities must be created to populate the virtual world. Entities are such things as tanks, personnel carriers, surface ships, aircraft, submarines and even individual combat soldiers.

### **1. Littoral Warfare and the Submarine**

The US Navy is rethinking its war-fighting doctrine, shifting from the blue-water Navy of the past to the littoral war fighters of the future [Dalton94]. A blue-water Navy prepares to do battle in the open ocean against another navy that is also intent on fighting at sea. Littoral warfare, however, takes place on the waterways close to land, the land close to water, and the airspace above the sea-land border. Unfortunately, most of the U.S. Navy's training has been for blue water conflict. As a result, Navy personnel in general know little about the science of littoral warfare. One expensive way to rapidly increase awareness and knowledge about littoral warfare is to stage large scale joint battles in various littoral regions with all the appropriate branches of the U.S. armed forces. A more cost-effective method of education is to simulate littoral warfare using a distributed virtual environment such as NPSNET.

NPSNET makes it more practical to invite participants from various warfare communities into cooperative training exercises. All can gain valuable experience and have their contributions included from different parts of the country. Using NPSNET, a costly antisubmarine warfare (ASW) exercise can be run without any of the combat participants leaving home port Figure 1.[Schmidt93]

One important platform in littoral warfare is the submarine. A submarine has the ability to move close to shore without revealing its presence. From this vantage point, a submarine is an excellent vehicle for running reconnaissance missions, stealthily transporting SEALs for penetration operations, laying minefields, or launching a rapid, highly accurate strike using Tomahawk missiles.



**Figure 1: Virtual Anti Submarine Warfare Exercise**

## **2. Teamwork vs. Individual Training**

Systems like NPSNET have been developed to help train groups of individuals, each of whom controls a separate vehicle. However, to operate many vehicles currently used by the armed forces require a group effort. Several individuals must work together to correctly control and employ the vehicle. This is true for tanks, bomber aircraft, ships and submarines.

Today, a handful of simulators let a group control the same entity, but NPSNET is not one of them [Sullivan93]. Cooperation is a critical part of the training process, as team members must work together. Prior to this thesis, NPSNET did not have the capability to let more than one person participate in the control of a vehicle. Since NPSNET is already

a distributed system, it is possible to add the technology needed to allow for various individuals to participate in the control of a single entity from multiple locations.

## **C. OBJECTIVES**

The objective of this research is to design an accurate submarine vehicle and incorporate it into NPSNET-IV. To achieve this objective, the submarine requires the following capabilities:

- The motion must be physically based, both underwater and on the surface. This allows the vehicle to be used as a trainer for junior officers. They can "drive" the virtual ship and get a feel for how a submarine would actually respond.[Hearn93]  
[Nobles95]
- The submarine must be operable by one or more individuals at the same time. An actual submarine is not an individually controlled machine, it requires teamwork to employ one effectively. If the virtual submarine is to be used for training, it should train a team, not simply an individual. Additionally, the controls must remain relatively simple so that a single person is capable of operating the submarine if no others are available.
- The weapons of the submarine must look and behave similarly to the normal weapons of a submarine, (e.g. the Mk 48 or ADCAP torpedoes, and the Harpoon and Tomahawk missiles.) Though the weapons should resemble the actual ones, they ought not be modeled exactly after the actual weapons, since this would lead to classification problems.
- Since NPSNET is distributed openly, all of the source code is unclassified. The submersible hydrodynamics coefficients must be unclassified, yet the model must have the capacity to quickly change to classified coefficients. This is accomplished through the use of parameter files that are used to initialize the hydrodynamic model. Classified coefficient files may be easily substituted with no other changes to the code.



## **D. THESIS OUTLINE**

The previous sections of this chapter state the objectives and motivation for providing a submarine vehicle for NPSNET. Chapter II provides background of the NPSNET project including purpose, history, general program design and current research. Chapter III discusses the design considerations of the hydrodynamic model that will control the submarine, including general hydrodynamic theory, survey of earlier work, derivation of a real-time buoyancy model, generation of representative unclassified hydrodynamic coefficients, and computational complexity considerations. Chapter IV describes the requirements and implementation of a protocol that allows more than one person to control a single vehicle. Chapter VI covers the graphical ocean environment required for the submarine to operate in the virtual world of NPSNET. Conclusions reached in this research, including run-time performance, limitations of the submarine and suggested future work, are in Chapters VII and VIII.



## **II. RELATED WORK**

### **A. NPSNET**

Students in the Naval Postgraduate School's Department of Computer Science began a project known as the Naval Postgraduate School Networked Virtual Environment (NPSNET) in 1990. NPSNET was written as a real-time, networked software package running on commercial off-the-shelf workstations, i.e. the Silicon Graphics Incorporated (SGI) IRIS family of computers. NPSNET was originally envisioned as a low-cost, government owned, workstation-based visual simulator [Zyda94]. NPSNET-IV.8.2 is the current version of the evolving NPSNET simulation system. It uses the Distributed Interactive System (DIS) Protocol version 2.0.3 for networked communications, and follows the object-oriented programming paradigm for defining and controlling remote and local DIS-based entities and munitions [Barham94].

DIS defines twenty-seven standard Protocol Data Units (PDUs), or information packets, for sharing information between simulators. Currently, NPSNET uses only three of the PDUs: Entity State, Fire, and Detonation. These three PDUs are the only ones required for basic interactive simulations. The other twenty four PDUs are relevant mainly in large, high-fidelity military exercises, or are not critical for the virtual world research currently being conducted at the Naval Postgraduate School[Barham94]. A notable deficiency is failure to implement the Message PDU [Brutzman94]. Currently NPSNET has implemented an Interface Data Unit (IDU) along with a IDU manager. IDU's can be specified prior to runtime, and used to communicate entity control data across the network.

Networked Virtual World simulation systems sharing the same Virtual Environment must have the capability of sharing information about the virtual environment, transmitting the events the local simulator causes and receiving events caused by remote entities in the virtual environment. This information includes such 3-D data as entity position and orientation (collectively referred to as posture), weapons firing, tracking and detonations. In a DIS virtual environment, each simulator maintains its own world

database copy of the objects in the world (e.g. terrain, buildings, trees, etc.) and is responsible for managing the local copy by keeping it up to date as the simulation progresses.[Barham94]

NPSNET IV uses a Euclidean coordinate system to specify an entity's posture in the virtual world. As stated, a posture is composed of a position specification, in this case (x, y, z) and a orientation specification (heading, pitch and roll). These three axes, combined with orientation displacements, enable an entity to obtain six degrees of freedom for motion. Linear and angular velocities and acceleration can also can be passed via PDUs to enable remote users to dead reckon entities between PDUs. Dead reckoning is defined as the process of calculating a new posture for an entity with no additional inputs. This is done by taking the initial posture and adding to it displacements that are equal to the respective velocity multiplied by the time since the initial posture was received.

In order to keep track of remote or networked entities, the local simulator must have some type of entity list containing entity identification, posture, geometry model data and other pertinent information. One way of reducing network traffic in a DIS simulation is to require the local simulation system to dead reckon each remote entity's position between the intermittent receptions of actual Entity State PDUs. This dead reckoning implies that the data concerning remote entities maintained in the entity list is lower resolution compared to the higher resolution knowledge of the locally-controlled entity.

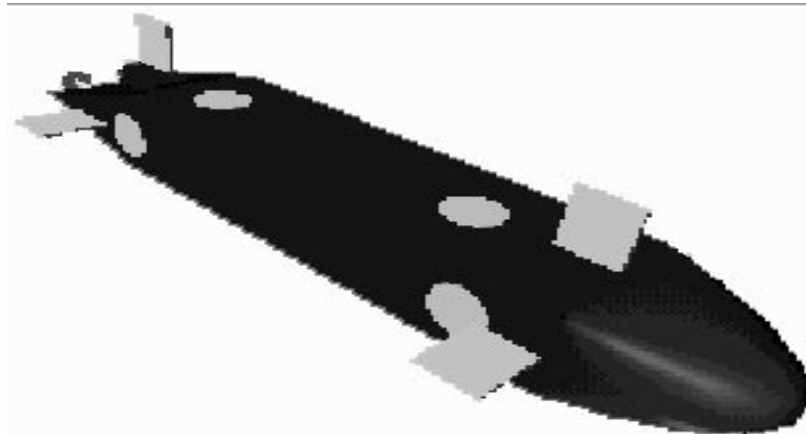
## **B. HYDRODYNAMICS**

There have been a few attempts to create submersible vehicles in virtual worlds. The first one developed at the Naval Postgraduate School took an NPS Object File Format (NPSOFF) submarine object and animated it under the constraints of accurate hydrodynamic laws of motion [Juerwicz90]. Unfortunately the physically-based modeling representation of the dynamics in this project is hard coded for one specific underwater vehicle. Adding or adjusting the submarine dynamics is not a simple task, and the

integration of a physically different submarine model requires software maintenance by a knowledgeable programmer [Zyda91] and rederivation of equations of motion by a mechanical engineer. Because of these problems, this work is unsuitable for general use.

A Deep Submergence Rescue Vehicle (DSRV) was also modeled using a simple Newtonian force-based paradigm [Zehner93]. As before, the model used was not general enough to allow for directly changing vehicle characteristics, such as length, width and buoyancy. In addition, the networking attempted used only simplistic stubs to provide a proof of concept. Benefits of this model included an object-oriented structure for physically based modeling.

In 1994, a rigorous general model for submerged vehicle hydrodynamics was created that was computationally suitable for real-time simulation [Brutzman94]. This model was developed to support computer simulation and testing of the NPS Autonomous Underwater Vehicle (NPS AUV) project Figure 2. This is the first publicly available hydrodynamic model that is based on standardized equations of motion and operates in real-time (10 Hz or faster control loop.) Another advantage of this model is its object oriented design, which allows programmers to implement and adjust for various types of submarines.



**Figure 2: NPS Autonomous Underwater Vehicle (AUV)**

The initial implementation of this model is that it assumed the submarine remained submerged at all times, hence the buoyancy of the submarine remained neutral. Based on

results reported in this thesis, the model now allows for real-time changes in a submarine's buoyancy, and includes a function for buoyancy versus depth. This functionality allows the AUV to surface and act much like a surface ship. This hydrodynamics model may conceivably be used to provide even more realistic training for other ship simulators [Hearn93][Nobles95].

### **C. MULTI-PLAYER CONTROL**

There are many different types of vehicle simulators in existence. Some simulators have the capability to train small groups of people to control a single vehicle, such as airline cockpit simulators that train a crew consisting of a pilot, co-pilot, and navigator together. Other simulators allow individuals to each control one vehicle, network the individual vehicles together and train as a group. No system exists however, that allows individuals to work as a small group controlling a single vehicle while at the same time remaining networked at different locations. NPSNET provides the required basic networking protocols and allows for rapid implementation of physically based models to be incorporated into the virtual environment. NPSNET and the DIS protocol, however, currently do not provide a means for a single vehicle to be controlled by more than one operator. Although the DIS protocol might be utilized to achieve this functionality using Message PDUs, such work has not been done.

A step in this direction for NPSNET was thesis work that created a remote panel from which a single individual user could control a vehicle [McMahan94]. The protocol used the basic NPSNET architecture coupled with a DIS-like protocol. This allowed a remote workstation to send a message containing the control information for one vehicle to another workstation (or host) that was simultaneously running the NPSNET simulation. The host workstation sends back to the remote controller updates of the vehicle's current posture, speed, sensor information, etc. However, this protocol does not have any means to direct these "control" and "information" packets to more than one machine. Because of this,

only one remote site can be operated at a time. Also, there is no hierarchy for means of control. If more than one remote station tries to send a "control" packet to the host, a race condition can occur where the host does not know which packet is the correct one to use. Multi-player control requires the ability to address "control" and "information" packets. It also requires a control protocol which prevents race conditions with the host workstation. Finally, authentication of control packets must be considered in order to prevent unauthorized interference.

#### **D. GRAPHICAL REPRESENTATION OF AN OCEAN ENVIRONMENT**

Providing a more realistic visual ocean is a crucial part of making an accurate submarine model. Current submarines use their periscopes infrequently, usually relying on passive sonar to determine locations of contacts. Because of this, officers aboard naval submarines get little practice looking through periscopes. The periscope, however, is one of the most vital sensors for a submarine. It allows reconnaissance of coast lines, obtaining visual navigation fixes, and provide rapid solutions to surface Target Motion Analysis (TMA) problems. Most shore-based periscope trainers use unrealistic flat oceans and toy contact models, blocking out the view with a single blue polygon occasionally to simulate actual ocean waves. Incorporating a realistic open-ocean environment adds more realism, thus immersing the individual further in the Virtual Environment, increasing the user's sense of presence and adding to the training value [Covington94].

The current open-ocean theater implemented for NPSNET does not allow for use in a virtual world representing a littoral region [Covington94]. One reason for this is the intersection of nonliving polygons (representing land) with moving polygons (representing wave motion) is difficult to calculate. Another problem with representing large bodies of water is that it is too computationally expensive to model waves across the entire ocean. This is an excellent area for future work.





### **III. HYDRODYNAMIC MODEL**

#### **A. INTRODUCTION**

A physically based hydrodynamics model is a set of mathematical equations that simulates the combined forces on a submerged rigid body. This model must be able to update a vehicle's posture, velocities and accelerations in the six degrees of freedom as discussed in the previous chapter. These updates must account for the complex physics of motion through water.

The effects of forces and moments can all be cross-coupled between vertical, lateral and horizontal directions. The effects of the surrounding environment are relatively large and significant, so much so that the adjacent water tends to be accelerated along with the vehicle and can be thought of as an "added mass." Together these challenges make underwater vehicle physical response, guidance and control an extremely difficult dynamics problem. [Brutzman 94]

Because of these challenges, the underwater hydrodynamics problem is too complex for a meaningful kinematics (velocities-only) solution [Healey93]. A more appropriate solution is to model all forces acting on the submersible and calculate the corresponding accelerations. Until this recent result, no single hydrodynamic model was general or robust enough to provide accurate information about all aspects of hydrodynamic motion while at the same time providing the means to vary pertinent inputs into the model. Furthermore, the ability to change the hydrodynamics coefficients allows one to model any form of submersible.

Reflecting the complex nature of underwater hydrodynamics, hydrodynamics models use more than one hundred pertinent coefficients. These coefficients can be obtained through costly analysis along with tow tank modeling. Most of the coefficients however, are of negligible importance to the physical behavior of a large underwater vehicle since they deal with second and third order effects.

The NPS AUV hydrodynamics model is rigorous, while enabling the programmer to easily change coefficients and dimensions of the submersible being modeled. This model provides six degrees of freedom and real-time performance in computer-based visual simulation [Brutzman94].

## **B. BUOYANCY FUNCTION**

### **1. Buoyancy in General**

As currently implemented, submersibles in the hydrodynamics model are considered neutrally buoyant. That is to say, the weight of the water which the submersible displaces is equal to the weight of the submersible itself. To enable a submarine to have a functioning ballasting system and to enable surface operations, the hydrodynamics model must be extended further.

Archimedes' principle states that a submerged body is subject to a buoyancy force that is equal to the weight of the fluid displaced by the volume of that body. The point at which all forces producing the buoyant effect may be considered to act is the center of buoyancy (CB) and is the volumetric center of the fluid displaced. For a stable floating object, the center of buoyancy is directly above the object's center of gravity (CG). The center of gravity is, for any three-dimensional body, the point where the net force related to the body's weight, or mass, may be considered to be located. Failure to put the center of buoyancy over the center of gravity results in static instability and a tendency to invert. To determine a vehicle's buoyant force we use the following formula:

$$\text{Buoyancy} = \rho g \iiint dV$$

Where:  $\rho$  is the density of water

$g$  is gravity

$\iiint dV$  is the volumetric displacement at a given time

This formula determines how much water volume a vehicle displaces at a given time and then multiplies this integration by the density of water and gravity. Although this

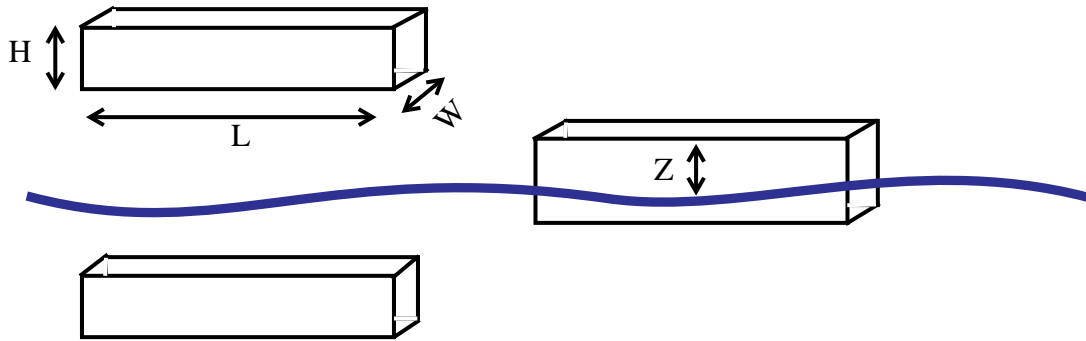
is the general solution for buoyancy, it involves a triple integral. This is deemed too computationally expensive for real-time use, since a submersible is not only an irregular shape, but the volume displaced changes rapidly as it moves up and down at the water's surface. A simple solution to this problem is to use a shape with an easily calculable volume that resembles the submersible. Although this is not precisely accurate, it models the major portions of buoyancy and is rapid enough for real-time application. Furthermore, such a simplification is correct at bounding conditions corresponding to neutral buoyancy. Thus any inaccuracies are only due to transient approximations. These approximations are insignificant compared to the variability already present due to surface wave action.

## 2. Function Based on Depth

Let us consider a rectangle as a simple vehicle with length  $L$ , height  $H$ , and width  $W$ , as in Figure 3. When the vehicle is not in the water, the vehicle has no buoyant force. When the vehicle is fully submerged, the buoyant force is equal to the volume of the vehicle  $L * H * W$ . With motion only in the vertical direction, the vehicle's buoyancy can quickly be determined by calculating the volume submerged.

$$\text{Volume} = \text{Length} \times \text{Width} \times (\text{Height} - Z)$$

Here,  $Z$  is the amount of "freeboard" or distance from the water line to the level deck of the vehicle.

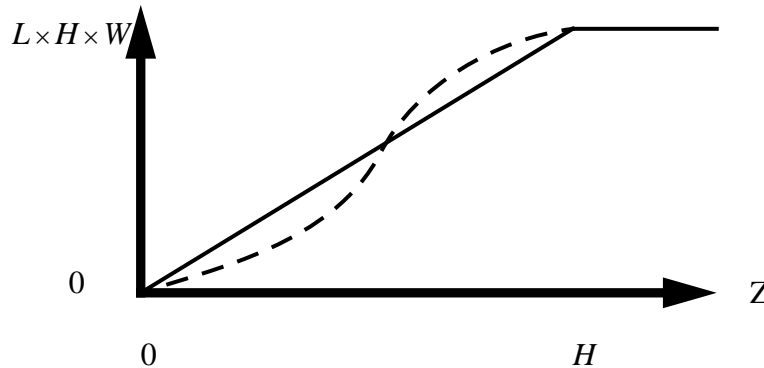


**Figure 3: Buoyancy related dimensions.**

As seen in Figure 4, when the vehicle is not in the water at all, buoyant force is equal to zero. As the volume of the vehicle enters the water the buoyant force increases until the entire volume is submerged. For a surface ship with a given mass, if the buoyant force is

less than its mass, it will sink into the water. Likewise, if the surface ship is too deep in the water, causing the buoyant force to be larger than the mass of the surface ship, it will move towards the surface of the water. This simple example demonstrates that a ship will tend to move to a point at which the surface ship's mass is equal to the buoyant force.

Buoyancy (lbm or kg)



**Figure 4: Buoyancy versus keel depth for a rectangular box (solid line) and a cylinder or ellipsoid (dotted line)**

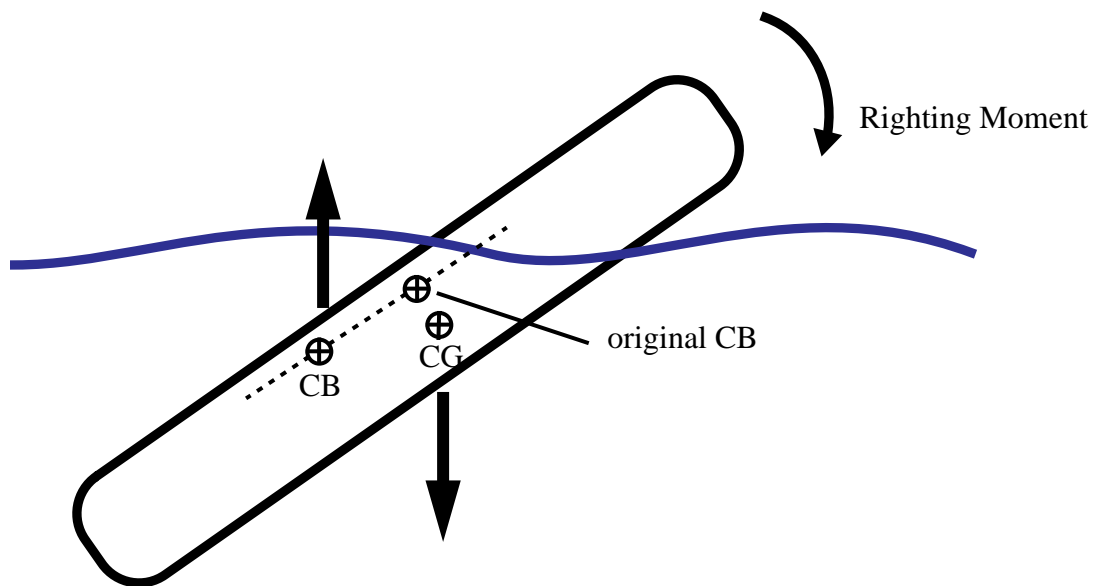
A submarine is not a rectangle; it more closely resembles a cylinder or ellipsoid. Its buoyancy function is bowed in the beginning due to less volume entering the water, grows steepest when the widest part of the ship is entering, and again tapers off as the top of the cylinder is reached. Figure 4 also shows that the change in buoyancy when using a cylinder versus a rectangle is a physically small affect that will only effect transient response, not bounding conditions or steady-state behavior.

### 3. Buoyancy Function Based on Ship's Angle

The previous discussion was based on a vehicle that changed displacement only in the vertical direction and did not consider pitch or roll. Since a submarine is almost perfectly symmetrical along the longitudinal axis, roll does not play a significant part in determining buoyancy. Such is not the case for a submersible's pitch. Submersibles have the capability to achieve large pitch angles of up to forty-five degrees while submerged. When the submersible is near the surface, these pitch angles can have a major impact on the vehicle's buoyancy in two ways. First and most obvious, more of the submersible can

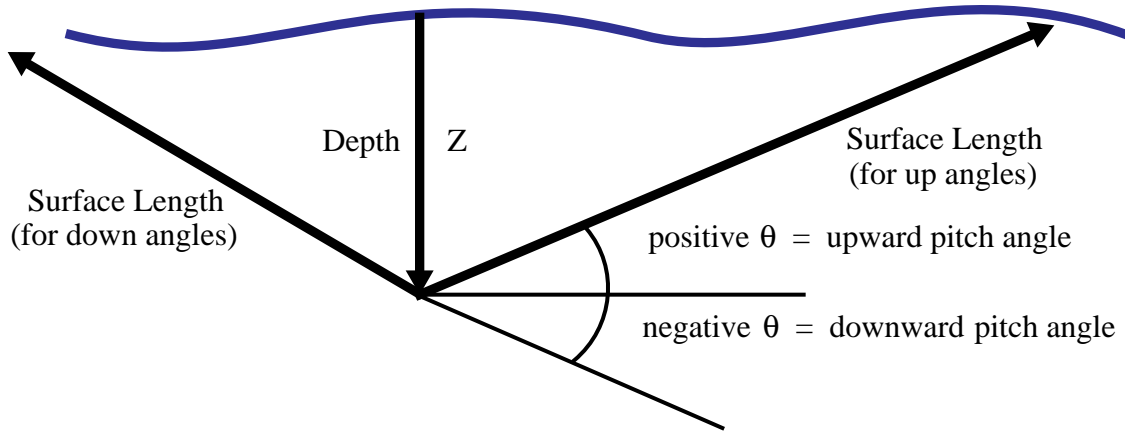
be moved out of the water. This of course tends to reduce the ship's overall buoyant force. The second and more significant impact is the change in the center of buoyancy. As stated, the center of buoyancy is the location where the buoyant forces act upon the vessel and is based on the location of the center of the displaced water. When a large portion of a submersible is out of the water, the volumetric location of the center of buoyancy moves to the center of the volume still submerged. If the bow is protruding the center of buoyancy will move aft, and if the stern is protruding CB will move forward.

Because of this shift in the center of buoyancy, a moment arm is created with the center of gravity. With the buoyant force pushing up from astern and the mass of the ship pulling down at the longitudinal center, a submersible tends to pitch down Figure 5.



**Figure 5: Moment arm created by change in CB position.**

In general, the distance that the center of buoyancy shifts is related to the volume of the submersible above the water. Volume above the water's surface can be simply estimated using only the pitch angle and centeline distance above the water. The centerline distance above water is also quickly calculated using only depth and pitch angle of a vehicle Figure 6.



**Figure 6: Surface Length determination.**

The centerline distance (CD) above the water is thus calculated with the following formula:

$$\text{CD above water} = \max((\text{Total Length} / 2) - \text{Surface Length})$$

$$\text{Surface Length} = \text{Depth} / \sin \theta$$

Furthermore, we must constrain this value to avoid inappropriate calculations when submerged.

$$-\text{Total Length} / 2 < \text{CD} < \text{Total Length} / 2$$

Combining the effects based on depth and pitch, a realistic representation of a submersible's buoyancy is obtained. Although this is a simplified buoyancy function, it demonstrates that the hydrodynamics model has the capability to adequately compute buoyancy changes under real-time restrictions.

## **C. IMPLEMENTATION**

### **1. New Coefficients**

As stated earlier, numerous coefficients are required for a hydrodynamic model to determine correct solutions. Hydrodynamic coefficients have been normalized with respect to vehicle length  $L$ , and thus are dimensionless quantities. These coefficients help describe

the relationships between the various velocities and acceleration of a submerged vehicle to the hydrodynamic effects that vehicle experiences.

Other important inputs to a hydrodynamic model are the measurements that define the physical layout of the submersible being modeled. Measurements include such things as length, mass, volume, cross-sectional areas of the submersible along with precise locations of propellers, planes and rudders. Currently the hydrodynamics model uses coefficients designed to correlate with the NPS Autonomous Underwater Vehicle (AUV). The AUV is an actual submersible six feet long and weighing 387 pounds. Unlike other hydrodynamic models, the coefficients and measurements in this model are located in a separate computer file. This allows for quick and easy changes by programmers.

To create a submarine in the virtual world of NPSNET, new coefficients were needed. To ensure this thesis remains unclassified, all coefficients and other measurements were obtained from unclassified documents. Of course, if the improved accuracy of classified coefficients were required in the future, it is a simple matter to substitute them. In the first iteration, the coefficients and measurements were obtained from the David Taylor Research Center, Ship Hydrodynamics Department, Departmental Report DTRC/SHD-1298-08 September 1990[Roddy90]. This report describes a 14.292 foot, 1556 pound general submarine form. A stable set of coefficients and measurements was obtained using this report and a series of validating laboratory missions. Then the general submarine model was ratioed up to a 360-foot, 6,900-ton submarine. This is a simplistic approximation corresponding to a Los Angeles class submarine[Sharpe93]. Both sets of coefficients and pertinent measurements are located in Appendices A and B, respectively.

## **2. Coordinate System**

To uniquely describe motion in a three dimensional world, a standard reference system must be established. Unfortunately, there is no universal frame of reference convention. Many related disciplines use conventions that are different from one another. This thesis relies on two conflicting references. NPSNET uses a different frame of

reference than the hydrodynamic model. These discrepancies must be resolved. The following describes the two frames of reference and how they were resolved.

***a. NPSNET***

NPSNET uses the convention of a right-handed three-dimension Euclidean space. The horizontal axis is the X axis, positive from left to right. The vertical axis is the Z axis, positive from bottom to top. The Y axis is perpendicular to the X and Z axis, positive from near to far. In global coordinate space, the Y axis is considered to point to a Northern compass heading, positive X pointing to the East, and positive Z indicating Height above mean sea level.

As is standard in most world coordinate systems, counter-clockwise rotations about the X, Y, and Z axes define the positive Euler angles phi, theta and psi ( $\Phi, \theta, \Psi$ ) respectively. These are commonly know as pitch, roll and heading. A vehicle's *posture* can be uniquely described given x, y, z coordinates along with  $\Phi, \theta, \Psi$  Euler angles.

***b. Hydrodynamics Model***

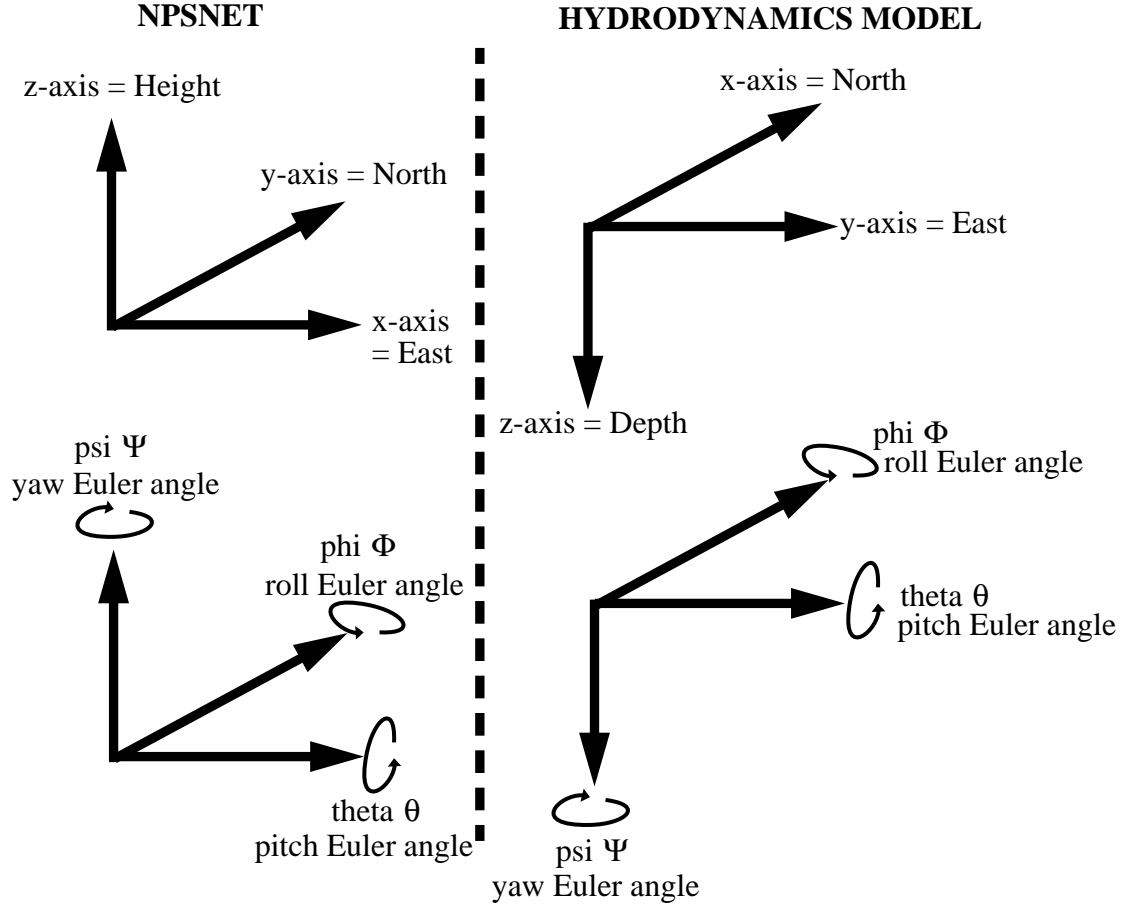
The hydrodynamic model also uses a right handed, three-dimensional Euclidean space but this coordinate system is based on the Naval Engineering standard. The horizontal axis is now the Y axis, positive from left to right, pointing to the East. The vertical axis is still the Z axis, but now positive has flipped, going from top to bottom. This is because the Z axis is now a measurement of depth. The X axis is now perpendicular to the Y and Z axis, positive from near to far, along the vehicle longitudinal axis, pointing towards North.

With a new coordinate system, roll, pitch and heading are defined again using right-handed counter-clockwise rotations about the three axis. Roll is provided from the X axis, pitch from the Y axis and heading from the Z axis.



### c. Integration of two reference models

Reconciling the differing reference conventions between the hydrodynamics model and NPSNET is straight forward when both conventions are viewed side by side Figure 7. The actual conversions are done within the local entity Update function of NPSNET.



**Figure 7: NPSNET and Hydrodynamics Model coordinate conventions**

Note that although roll and pitch map directly, heading is reversed since the Z axis has been inverted. Appropriate sign transformations for heading  $\Psi$ , depth Z and east coordinate Y are used to make the two references compatible.

### **3. Inputs and Outputs**

All vehicles must be provided some means of control as they operate in their native environment. Typically a submerged vehicle is outfitted with a propeller of some kind along with a combination of control planes. Control over weight and buoyancy is typically maintained through the use of a ballast and trim system.

Although propellers and thrusters may be positioned and oriented in various ways on a submersible, for simplicity the submersible modeled in this thesis has one main propeller. This main propeller is located aft along the centerline of the submersible and provides forward and aft thrust.

The main purpose of a control surface is to induce a moment on the vehicle to cause it to rotate to a desired angle. Control planes are usually aligned in either the horizontal or vertical direction and rotate around their axis of orientation. A rudder is a plane aligned in the vertical direction and is used to control the submersible's heading. Stern and bow planes are aligned in a horizontal direction and are used to control the submersible's pitch. When neutrally buoyant and moving forward, pitch control is via the bow and stern planes allowing the submersible to change depth.

## IV. MULTI-CONTROLLER PROTOCOL

### A. INTRODUCTION

As described in NPSNET-IV: An Object-Oriented Interface for a Three-Dimensional Virtual World, [McMahan94] a remote control panel for NPSNET was developed that has the ability to control a single NPSNET entity across a network. This is done using unique data structures passed over network socket connections between the machine running the remote panel application and the NPSNET host machine. The information sent between the two platforms allows for a loose coupling between the interface application and NPSNET. A benefit of this system is that changes can be made quickly to one application with few effects on the other [McMahan94].

Although successful as a remote communication application, McMahan's work was built with the paradigm of single user controlling a single vehicle. In order for McMahan's work to be useful in a multiuser application, it needs to be expanded to allow several users to control a single vehicle.

One problem encountered while extending McMahan's work directly into a multi-user control protocol was dealing with the "race conditions" that can ensue. A "race condition" results when two or more processes are reading or writing some shared data and the final result depends on the precise order of the processes executed. With any multiuser control protocol, race condition situations must be addressed. This problem becomes more apparent when the remote control application enables an individual user to assume the role of any of the possible control positions. The basic issue is that a remote control application must be prevented from sending erroneous control information to the main application, while at the same time it must be constantly up-to-date with the other remote control applications that are running.

## **B. SUBMARINE CONTROL PANELS**

In this thesis, the remote control application covers three user roles in the effort to operate a submarine—specifically a Helmsman, an Officer of the Deck (OOD), and a Weapons Officer. Each user controls different parts of the on-screen panel, which are cooperative elements of the same application. In addition, each user has the ability to change roles at any time during the simulation. The sub control application is flexible enough to support three users, while at the same time can easily be controlled by one individual. General layout and design of the three panels are based on making the controls intuitive while retaining functionality. [Schneiderman92]

### **1. Helm Control Panel**

The Helmsmen's responsibility in the operation of a submarine is to steer the submarine so that it remains on the depth and course that is ordered by the OOD. The Helmsmen's Panel therefore contains various indicators to tell him or her the current depth, course, and speed of the submarine, in both digital and analog readouts Figure 8. Along with these readouts are indicators that show the orientation of the various control surfaces of the submarine, rudder, fairwater planes and stern planes. Actual control of these planes is accomplished using a joystick or keyboard inputs. In addition to steering the submarine, the Helmsman relays the ordered speed to the engine room through the use of the engine order telegraph. The submarine's speed can be set using the engine order telegraph, throttle control or keyboard input. On a normal submarine, the OOD makes his orders to the Helmsmen verbally. Due to the distributed nature of this application verbal orders may not be possible because the OOD and the Helmsman may be located a great distance apart. The ordered course, depth and speed display on the far left side of the Helmsmen panel fulfills this communication requirement.

### **2. Officer of the Deck (OOD) Control Panel**

The OOD's main responsibility is setting a path for the submarine to maneuver through the virtual world. To do this, the OOD control panel must have the ability to give

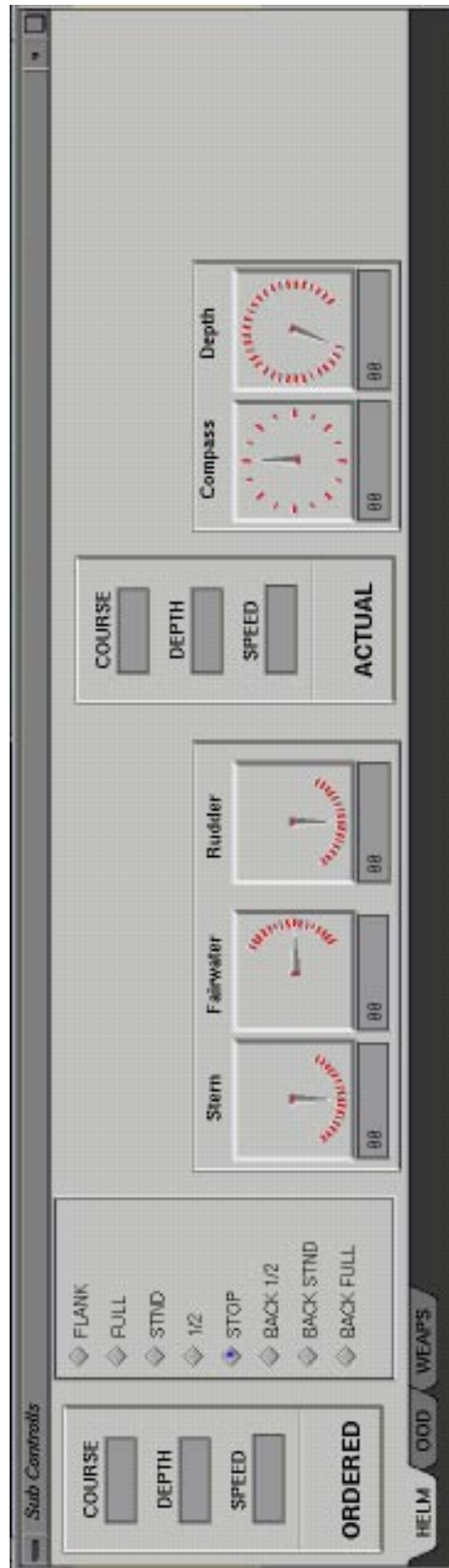


Figure 8: Helm Control Panel

orders and view the surrounding environment Figure 9. Orders are relayed to the Helmsmen through the use of the panel's ordered course, depth and speed areas. These orders can be monitored using the actual course, depth, and speed display. In addition to these orders the OOD has the Main Ballast Tanks (MBTs) controls.

The MBTs are empty while the submarine is on the surface, providing positive buoyancy. To submerge in an actual submarine, the MBTs vents, located at the top of the tanks, let water fill the MBTs from open grates in the bottom of the tanks. This in turn causes the submarine to lose its positive buoyancy and the submarine submerges. To surface, this process is reversed. The MBTs vents are shut and air is injected into the tanks using the Emergency Main Ballast Tank (EMBT) Blow system, forcing the water out of the open grates in the bottom of the tanks. In this virtual submarine, controls for both the MBTs vents and EMBT Blow system are incorporated into the OOD control panel.

To allow the OOD to view the environment, periscope controls are provided. These allow the OOD to raise and lower the periscope and change the elevation as well as the azimuth of the viewing direction. The periscope view also provides the ability to determine a bearing and range to an object that is being viewed by placing the object in the cross hairs of the periscope and then depressing the mark button. Bearing and range information is then displayed below the viewing controls.

### **3. Weapons Officer Control Panel**

The Weapons Officer is responsible for the launching of weapons, in this case unclassified versions of MK 48 or ADCAP torpedoes and Tomahawk cruise missiles. To carry out these functions, the Weapons Officer control panel has one area for torpedo controls and one for cruise missile controls Figure 10. Torpedoes are used to conduct Anti Submarine Warfare (ASW) or Anti Surface Warfare (ASUW). Tomahawk cruise missiles are used to conduct precision strikes against land targets.

The torpedo controls enable the Weapons Officer to select a tube to be fired, set a course for the torpedo to follow, then shoot the torpedo. Unlike torpedoes, the Tomahawk



Figure 9: OOD Control Panel

missiles are preloaded with flight paths and therefore no course needs to be set for these missiles.

### **C. COMMUNICATION PROTOCOL**

Revising the code to allow for multi-user control required labeling each of the control data packets sent from the remote control applications to the NPSNET host application. The labels identify the role of the user who generated the input. These labels allow the NPSNET application to parse out the appropriate information for a given label while ignoring the other data fields. For example, when a control data packet labeled “Weapons Officer” is transmitted to the NPSNET application via the socket connection, only the positions of the dials and inputs under the direct authority of the Weapons Officer will be registered and updated, such as torpedo tube selected, torpedo course and shoot commands. The controls not under the authority of the Weapons Officer, such as Helmsmen and OOD functions, will not be registered by NPSNET therefore eliminating the need for direct communications between remote control applications.

Once the NPSNET application has registered the useful information from an incoming control data packet, the application creates an updated master communication packet. This master communication packet contains the most current setting for all three control positions. NPSNET then sends this master communication packet to all of the remote controllers, so information is updated on the three control panels on all remote applications. That way, all control panels on each remote application stay in sync, and the numbers and settings displayed remain constant until the appropriate user intentionally revises them.



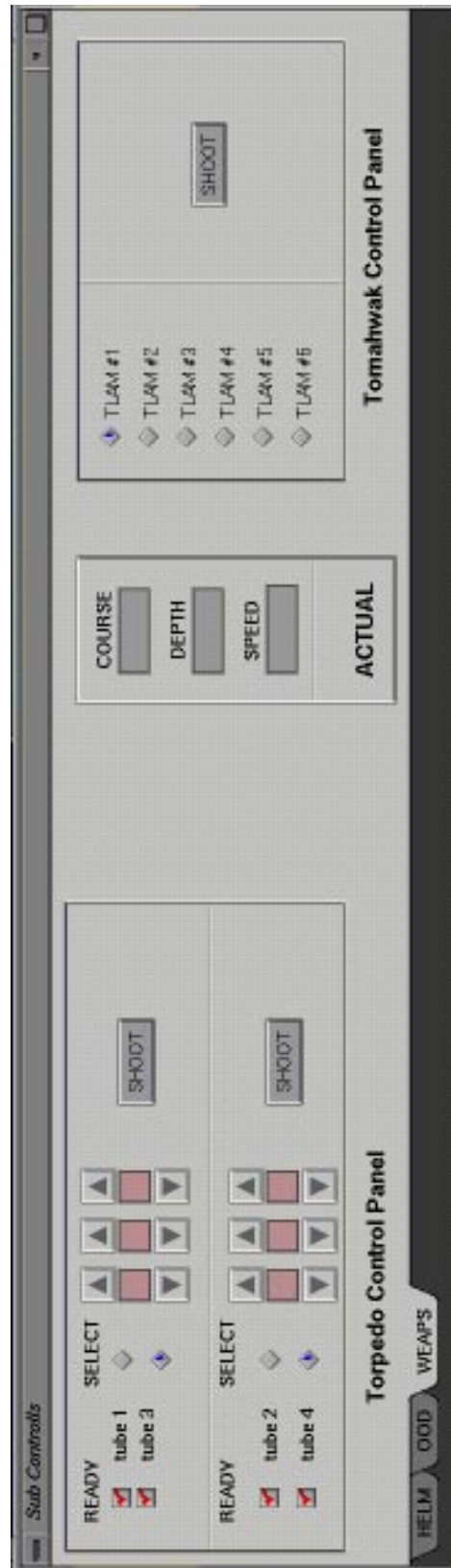


Figure 10: Weapons Officer Control Panel



## **V. GRAPHICAL REPRESENTATION OF OCEAN ENVIRONMENT**

### **A. INTRODUCTION**

Databases used in virtual worlds can be broken down into two major categories: the two-dimensional plane and the three-dimensional objects. The two-dimensional plane can be thought of as a table top or playing field and usually represents the surface of the land or large bodies of water. This plane defines the boundaries for the x and y axis of the virtual world. The next major category is composed of three-dimensional objects, buildings, vehicles, mountains, bridges and foliage that populate the virtual world. These objects can be thought of as being placed onto the initial two-dimensional plane. Because of this paradigm, the virtual world usually only extends into the positive third-dimension, namely height above the two-dimensional plane. One flaw with this system is that whenever a user moves below the flat two-dimensional plane of the virtual world, for example when an entity dives below the surface of the ocean, there is nothing there. It appears that the virtual world is suspending in air, allowing the user to quickly determine the unreality of the situation. Developing a simple means of creating an entire environment underneath this plane was required for this project, since submarines operate in this previously undeveloped region of the virtual world.

Limitations on the number of polygons per second a computers can render is diminishing with introduction of faster processors, specialized graphics generators and larger graphics pipelines. Computer graphics still have limits to their capabilities nevertheless. To ensure rapid frame rates are maintained to provide smooth realistic motion, polygon conservation when creating an environment is essential. When developing virtual worlds, programmers attempt to render as few polygons as necessary to achieve a graphically believable environment. One way of accomplishing this is using only a few large polygons in large flat areas of the virtual world, such as fields and oceans.

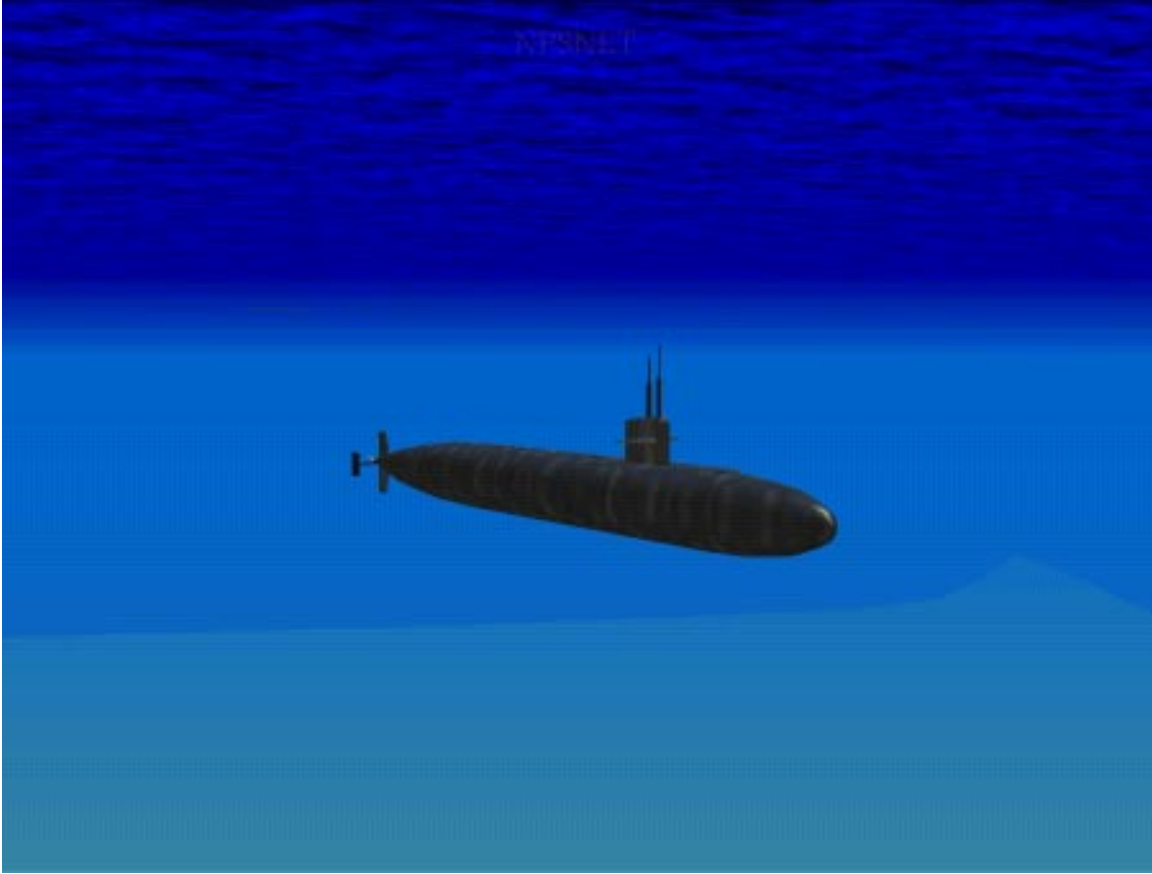
Challenges occur when these large areas need to appear as if in motion. Since it is impossible to bend a polygon, these areas must be created with several polygons, so that the surface looks to remain smooth throughout its "movement."

Programmers use textures as another way of minimizing the number of polygons in a virtual world. Textures are pictures that can be applied to a polygon, giving it a much more complex appearance. A simple example of this is a cube with textures applied to each surface that are pictures of the outside of a house. Instead of using individual polygons to model each door, window or brick, this texture allows a developer to place that rich, complex image in the virtual world using only five polygons (four walls and a roof). In NPSNET, this technique has been useful in creating buildings, trees and a realistic-looking ocean surface. Previously only a random noise texture was placed on the blue polygons that represent the ocean, thus giving the effect of ripples and waves as viewed from a distance. Unfortunately, as the viewer gets closer to these textured polygons, the optical illusion is destroyed and the "flatness" of the ocean becomes apparent.

## **B. GRAPHICAL UNDERSEA ENVIRONMENT**

To create an environment below the two-dimension playing field of the current virtual environment, two steps were taken. First, a new two-dimensional playing field was created. This new field is lower than the original one, representing not the ocean surface but the ocean bottom. Second, the space between these two planes was filled with a new graphical environment, namely the ocean water Figure 11.

The ocean bottom was created by taking a copy of the ocean surface polygons and adjusting all vertices along the negative z axis to an appropriate depth. Since the original ocean surface is represented by a group of flat polygons with a constant depth, the ocean bottom, in turn, now has a constant depth. The main purpose for creating an ocean bottom was to give the user a visual frame of reference when driving the submarine underwater. Without this frame of reference, up, down, left and right can quickly become confused. A



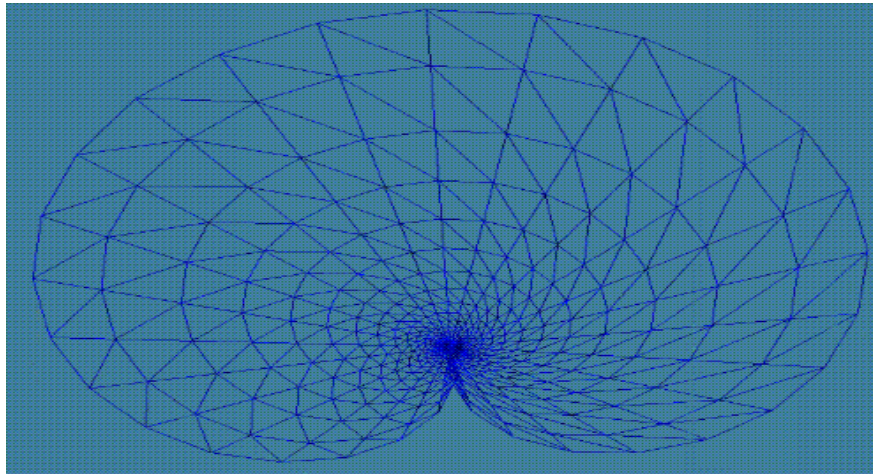
**Figure 11: Undersea Environment**

simple shoaling ocean bottom was deemed unimportant for the simulation and might make the ocean bottom more difficult to discern as a visual reference. Future work will undoubtedly apply bathymetric data to contour the ocean bottom.

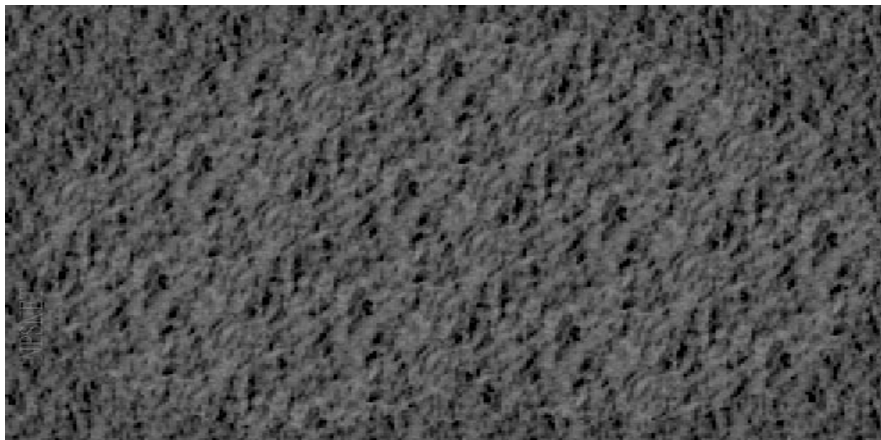
To create the effect of water, the Performer fog function was used. This function allows the programmer to specify the range, color and level of the fog, which is then applied to the scene. In this case, the fog is colored light blue with visual perception further impaired as depth increases. This fog is generated whenever the height  $z$  axis is below the surface of the water. An added benefit of using this fog is just as vision is difficult under real water, vision in this virtual world also is difficult. This also supports the argument that depicting a shoaling ocean bottom is not significant.

### **C. OCEAN WAVE CARPET**

Although submarines mainly remain submerged, the rendering of wave motion is essential to provide realism while the submarine is near the surface with its periscope in use. Two principles are critical when considering how to create surface wave motion. First, the closer objects are, the more realistic they must look. Second, making the entire ocean out of moving polygons can easily become too complex to be practical or possible. Thus, a moving "ocean carpet" was developed. This carpet consists of 824 polygons suspended one meter above the "flat" ocean polygons Figure 12. The ocean carpet is colored and textured to look the same as the rest of the distant-ocean polygons Figure 13. The ocean



**Figure 12: Ocean Carpet (Wire frame)**

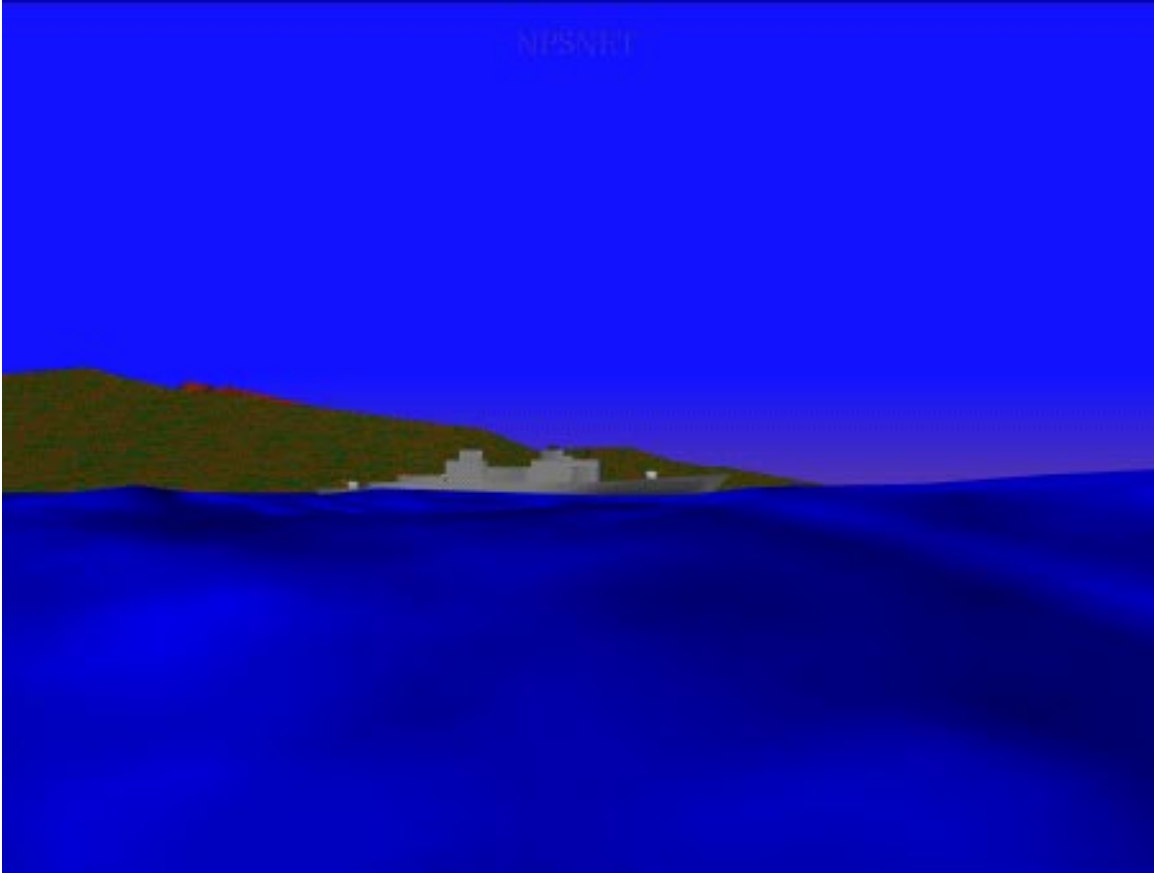


**Figure 13: Ocean Carpet (Textured)**

carpet polygons are animated so that they appear to move in a wave-like fashion. This animation takes advantage of the Performer graphics pipeline, using an array of three sets of vertices of the ocean carpet. Experimentation using only one set of vertices in the animation process was initially adequate, however three sets provided for truly life-like smoothness of motion.

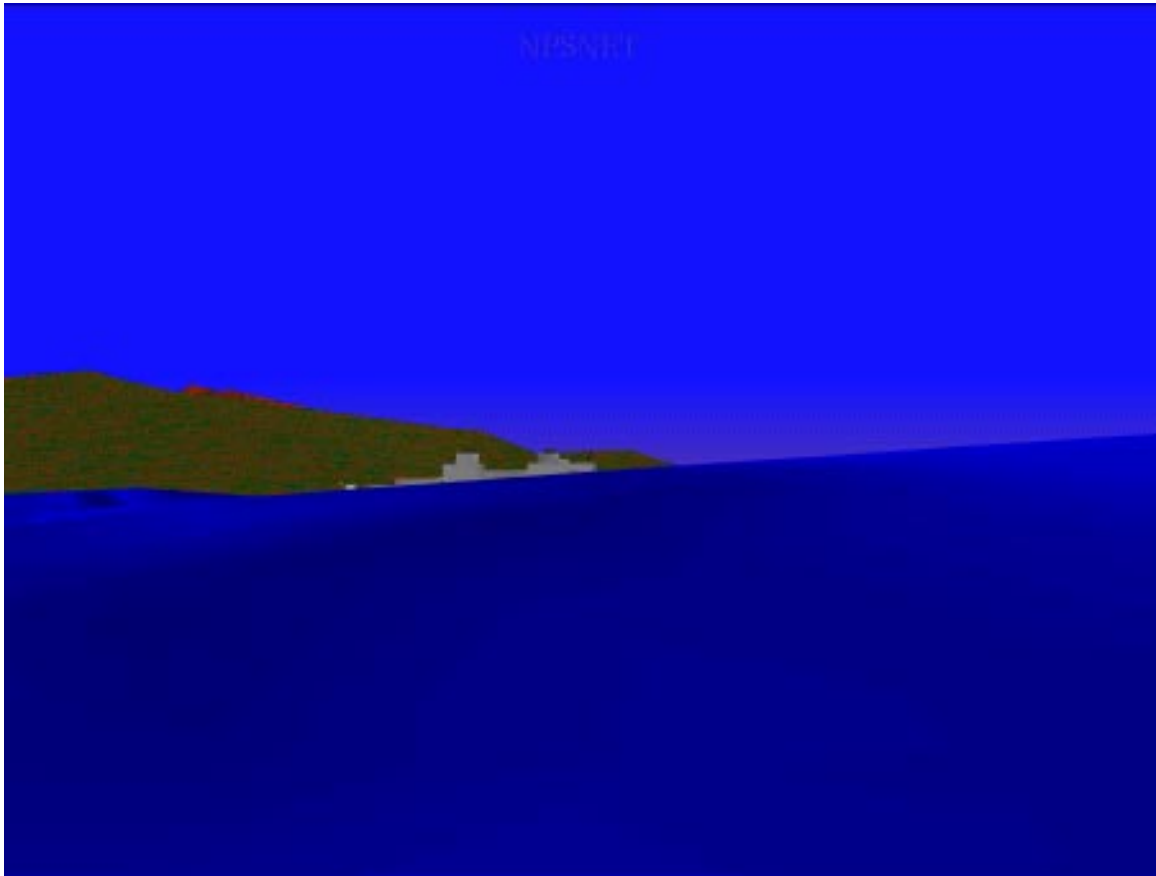
This carpet is positioned in the x and y axis and rotated around the z axis corresponding with the heading of the local entity. The ocean carpet therefore, follows the vehicle wherever it goes in the virtual undersea world while always remaining suspended one meter above the normal ocean polygons. Because the ocean carpet and the rest of the ocean are colored and textured identically, the edge of the carpet cannot be discerned from the user's vantage point. Similarity of texturing and color, coupled with the fact that the original texture has always been a good representation of waves from a distance, provides views from periscopes that are extremely realistic Figure 14 Figure 15 Figure 16.

This ocean carpet is not represented for any entities other than the locally generated one. This is because wave motion is only apparent close up, and in general most entities are so far away that the generation of waves around them becomes visually insignificant.



**Figure 14: Time Lapsed View from Periscope with Ocean Wave Carpet**





**Figure 15: Time Lapsed View from Periscope with Ocean Carpet**



**Figure 16: Time Lapsed View from Periscope with Ocean Wave Carpet**

## **VI. CONCLUSION AND RECOMMENDATIONS**

### **A. RUNTIME PERFORMANCE**

#### **1. HYDRODYNAMICS MODEL**

Physically based dynamic modeling, when properly parallelized, is less processor-intensive than graphics rendering [Juerwicz90][Zyda91]. In this application of a physical model, the hydrodynamic model used to control the submersible's motion was not parallelized. This led to no measurable degradation of performance in frame rates, maintaining approximately 30 frames per second. This experimental result confirms that the NPS AUV hydrodynamics model is suitable for real-time applications. Along with adequate frame rates, the hydrodynamic model, utilizing the new sets of coefficients, provided smooth accurate motion. Because of this motion, the overall experience of using the submersible is greatly enhanced.

#### **2. OCEAN WAVE CARPET**

Whenever more polygons are added to a scene, graphical performance may decline. Since the ocean carpet only adds 824 new polygons, there is not a significant performance decline in the average NPSNET scene. Along with the addition of more polygons requiring rendering, new positions of the various vertices for these polygons must be calculated for each frame. As stated in chapter V, this has been parallelized, taking advantage of the Performer pipeline. In addition, only one ocean carpet is generated to enhance only the local entity's view into the virtual world. If ocean carpets were to be generated for other entities, such as surfaced submarines or ships, there is a likelihood of performance degradation.

## **B. CONCLUSIONS**

The objective of this research was to enhance the training potential of NPSNET by implementing a submarine that has its motion based on a physically-based hydrodynamics model, and has controls that enable a individuals to use teamwork to control the vehicle.

After performing the development, testing and evaluation of the various aspects of this project, we have reached the following conclusions:

- Physically-based hydrodynamics models can be implemented in real-time.
- Control of a single entity by various distributed individuals can be accomplished using a multicast protocol.
- Generation of ocean waves in littoral regions can be accomplished in real-time.

## **C. LIMITATIONS**

As the current submarine implementation stands there are a few limitations to discuss. The most obvious one is that although a multi-controller protocol is in place for three individuals to operate the submarine, this task is normally done by over 20 crew members. To generate controls and supply individual workstations for this number is not feasible. This work implementation could readily be used for simpler platforms, not requiring as many operators, such as tanks, planes, an smaller vessels.

In addition to the scalability problem of controlling stations, there exists another scalability problem that was not addressed in this work. Since there is only one NPSNET process running, acting as the host to various controlling applications, there can be only one view into the virtual world. This view is not capable of being distributed at this time. This would be important if various participants needed to receive visual cues, like a pilot and co-pilot.

## **D. RECOMMENDATIONS FOR FUTURE WORK**

This work provides three major improvements to NPSNET IV: a physically based hydrodynamics model that can be quickly updated to represent various underwater vehicles; the capability to have multiple users control the same entity, improving the

training quality provided by NPSNET; and an improvement to the graphical representation of the ocean in NPSNET. Some areas of future work are:

- Extending the buoyancy function in the hydrodynamic model to incorporate the effect of wave motion.
- Modeling the wave motion based on standardized physical representation of waves.
- Integrate the NPS AUV control code into the NPSNET submersible vehicle class to allow scripted high-level behavior.
- Establish a way to let remote entities see moving ocean carpets around surfaced submarines and ships.
- Update undersea weapons with physically based movements.
- Extend the submersible vehicle software to create a graphical way to conduct Target Motion Analysis (TMA).
- Incorporating the ocean surface model into the Open Inventor-based NPS AUV undersea Virtual World.
- Integrate the extended hydrodynamics model into actual submersibles to provide a predictive self-modeling diagnostic capability.



## APPENDIX A - DARPA SUBOFF MODEL COEFFICIENTS

//

/\*

Program: SUBOFFmodel.H (Version of UUVmodel.H for DARPA SUBOFF model)

Author: Don Brutzman

Revised: 20 August 95

System: Irix 5.3

Compiler: ANSI C++

Compilation: irix> cp SUBOFFmodel.H UUVmodel.H

irix> make dynamics

Advisors: Dr. Mike Zyda, Dr. Bob McGhee and Dr. Tony Healey

References: Healey, Anthony J. and Lienard, David, "Multivariable Sliding Mode Control for Autonomous Diving and Steering of Unmanned Underwater Vehicles," IEEE Journal of Oceanic Engineering, vol. 18 no. 3, July 1993, pp. 327-339,

Lewis, Edward V., editor, \_Principles of Naval Architecture volume III\_, second revision, The Society of Naval Architects and Marine Engineers, Jersey City New Jersey, 1988, pp. 188-190 and 418-423.

Gertler, Morton and Hagen, Grant R., \_Standard Equations of Motion for Submarine Simulation\_, Naval Ship Research and Development Center (NSRDC) Research and Development Report 2510, Washington DC, June 1967.

Smith, N.S., Crane J.W. and Summey, D.C., \_SDV Simulator Hydrodynamic Coefficients\_, Naval Coastal Systems Center (NCSC), Panama City Florida, June 1978. Declassified.

Marco, David. "Slow Speed Control and Dynamic Positioning of an Autonomous Vehicle," Ph.D. dissertation, Naval Postgraduate School, Monterey California, March 1995.

Bahrke, Fredric G., "On-Line Identification of the Speed, Steering and Diving Response Parameters of an Autonomous Underwater Vehicle from Experimental Data," Master's Thesis, Naval Postgraduate School, Monterey California, March 1992.

Warner, David C., "Design, Simulation and Experimental Verification of a Computer Model and Enhanced Position Estimator for the NPS AUV II," Master's Thesis, Naval Postgraduate School, Monterey California, December 1991.

Bacon, Daniel K. Jr., "Integration of a Submarine into NPSNET," Master's Thesis, Naval Postgraduate School, Monterey California, September 1995.

Roddy, Robert F., "Investigation of the Stability and Control Characteristics of Several Configurations of the DARPA SUBOFF Model (DTRC Model 5470) from Captive-Model Experiments," Technical Report DTRC/SHD-1298-08, Ship Hydrodynamics Department, David Taylor Research Center (DTRC), September 1990 (unclassified).

Model note: We use values for DARPA SUBOFF model in Configuration 2 (fully appended) which includes plane surfaces.

Notes: const definitions are for software engineering reliability they can be changed to variables if coefficient modification becomes desirable



```
add clamp values for planes, rudders and propulsors
pass clamp values to execution level
```

////////////////////////////////////

//-----//

```
#define m4_ft4 (0.305)*(0.305)*(0.305)*(0.305) // (0.305 m/ft == 1)
```

```

; //      Inertia matrix coefficients
const double I_x  = 0.0    *m4_ft4 ; // Nms^2    = I_xx =
const double I_y  = 0.001053*m4_ft4 ; // Nms^2    = I_yy =
const double I_z  = 0.001084*m4_ft4 ; // Nms^2    = I_zz =
const double I_xy = 0.0    ; // Nms^2    = I_yx
const double I_xz = 0.0    ; // Nms^2    = I_zx
const double I_yz = 0.0    ; // Nms^2    = I_zy

#undef      m4_ft4

; //      Centers of Gravity & Buoyancy
const double x_G  = 0.556    *0.3048; // m
const double y_G  = 0.0      *0.3048; // m
const double z_G  = 0.0      *0.3048; // m      Note CG below CB  Marco 0.5"
const double x_B  = 0.532094*0.3048; // m
const double y_B  = 0.0      *0.3048; // m
const double z_B  = -0.006669*0.3048; // m      CB at center of UUV

//      Additional hull characteristics      //

const double H    = 0.240792 ; // m      main hull height 9.50"

double revisedBuoyancy, revised_x_B;

double surface_length = 0.0; // distances (CB to surface) & (CB to nose)
// along body axis
const double nose_length = (0.90 * L) / 2.0;

#else // (not SI) standard British units -----

const double Weight = 1556.2363 ; // lb      Weight
const double Buoyancy = 1556.2363 ; // lb      Buoyancy
const double L      = 14.2917 ; // ft      characteristic length

const double g      = 32.174 ; // ft/s^2      gravitational constant

```

```

const double rho    = 1.94 ; // slugs/ft^3 mass density of fresh water
const double m      = Weight / g ; // lb/ft-s^2 vehicle mass incl. free flood
                        // = 48.369376

// Moments of inertia units normalized using perpendicular length 13.9792 ft
// (Roddy p.3, Feldman p. 6)

#define Lnorm(i) (i * 0.5*rho*13.9792*13.9792*13.9792*13.9792*13.9792)

                                ; //      Inertia matrix coefficients

// new value I_x not found, verify through calculations ??
const double I_x      = Lnorm(0.000060); // lb-ft-sec^2      =I_xx

const double I_y      = Lnorm(0.001053); // lb-ft-sec^2      =I_yy
const double I_z      = Lnorm(0.001084); // lb-ft-sec^2      =I_zz
const double I_xy     = Lnorm(0.0)      ; // lb-ft-sec^2      =I_yx
const double I_xz     = Lnorm(0.0)      ; // lb-ft-sec^2      =I_zx
const double I_yz     = Lnorm(0.0)      ; // lb-ft-sec^2      =I_zy

// print values as comments here ***

#undef Lnorm(i)

// Model length = 14.292', body coordinate center (0,0,0) will be at
// midpoints (longitudinal = 14.292/2 = 7.146', vertical = 1.667/2 = 0.83')

// CG: x_G = 7.146 - 6.590 = 0.556'      (Figure 1 p. 9)
// CG: y_G = 0.0'                        (Figure 1 p. 9 not given)
// CG: z_G = 0.0'                        (Figure 1 p. 9)
// CB: x_B = 7.146 - 6.613906 = 0.532094' (Table 1c p. 14 LCB)
// CB: y_B = 0.0'                        (Table 1c p. 14 not given)
// CB: z_B = -0.006669'                  (Table 1c p. 14 VCB)

                                //      Centers of Gravity & Buoyancy
const double x_G      = 0.532094 ; // ft      listed as 0.556

```

```

const double y_G = 0.0 ; // ft

// *** modification - z_G moved down from 0.0 for more adequate righting arm ***
const double z_G = 0.5 ; // ft    Note CG below CB

const double x_B = 0.532094 ; // ft    0.010416667
const double y_B = 0.0 ; // ft
const double z_B = -0.006669 ; // ft

// Thruster/propeller distances from centerlines. Note stern/port are negative.

const double x_bow_vertical = 0.0 ; // ft No thrusters!
const double x_stern_vertical = 0.0 ; // ft
const double x_bow_lateral = 0.0 ; // ft
const double x_stern_lateral = 0.0 ; // ft

const double y_port_propeller = 0.0 ; // ft Single propeller, on centerline
const double y_stbd_propeller = 0.0 ; // ft

// Rudder bow/stern distances from centerlines. 0.5 is all the way forward/aft.

const double x_rb = 0.0 * L ; // proportional distance to bow (none!)
const double x_rs = -0.427 * L ; // proportional distance to stern

//      Additional hull characteristics      //

const double H = 1.667 ; // ft    main hull diameter 1.667 ft

double revisedBuoyancy, revised_x_B;

double surface_length = 0.0; // distances (CB to surface) & (CB to nose)
// along body axis
const double nose_length = (0.90 * L) / 2.0;

const int THRUSTERS = FALSE; // are cross-body thrusters present?

```

```

#endif

//-----//
//      Surge equation of motion coefficients      //

// *** warning: no X_ coefficients found in Roddy reference for ARPA SUBOFF
const double X_u_dot = 0.0 ;// Linear force coefficients acting in
const double X_v_dot = 0.0 ;// the longitudinal body axis
const double X_w_dot = 0.0 ;// with respect to subscripted
const double X_p_dot = 0.0 ;// motion components
const double X_q_dot = 0.0 ;//
const double X_r_dot = 0.0 ;//

const double X_uu = 0.0 ;//
const double X_vv = 0.0 ;//
const double X_ww = 0.0 ;//
const double X_pp = 0.0 ;//
const double X_qq = 0.0 ;//
const double X_rr = 0.0 ;//

const double X_prop = 0.0 ;// X_prop "constant" no longer applicable

// plane surface drags not given, either 0 or estimated at 1/2 AUV effectiveness
// (same swag factor for other coefficients later)

const double X_uu_delta_b_delta_b = 0.0 ;// drag due to bow plane
const double X_uu_delta_s_delta_s = -1.018E-2/2.0 ;// drag due to stern plane
const double X_uu_delta_r_delta_r = -1.018E-2/2.0 ;// drag due to rudder
// single plane/rudder set

const double X_pr = 0.0 ;// (these aren't in Bahrke thesis model)
const double X_wq = 0.0 ;//
const double X_vp = 0.0 ;//
const double X_vr = 0.0 ;//

const double X_uq_delta_bow = 0.0 ;//

```

```

const double X_uq_delta_stern = 0.0 ;//
const double X_ur_delta_rudder = 0.0 ;//
const double X_uv_delta_rudder = 0.0 ;//
const double X_uw_delta_bow = 0.0 ;//
const double X_uw_delta_stern = 0.0 ;//

const double X_qdsn = 0.0 ;// no longer used in new model
const double X_wdsn = 0.0 ;// no longer used in new model
const double X_dsdsn = 0.0 ;// no longer used in new model

// we assume 5 knot max speed = 500 yds/3 min = 500 ft/min = 8.333 ft/sec
// we assume max rpm is 200

const double speed_per_rpm = 8.333 / 200.0 ;// steady state: 0.04166
// = (8.333 feet/sec) per 200 rpm

const double MAX_RPM = 200.0 ;// single propeller only

// *** recheck this value:
const double C_d0 = 0.00778 ;//

//-----//
// Sway equation of motion coefficients //

const double Y_u_dot = 0.0 ;// Linear force coefficients acting in
const double Y_v_dot = -0.016186 ;// the athwartships body axis
const double Y_w_dot = 0.0 ;// with respect to subscripted
const double Y_p_dot = 0.0 ;// motion components
const double Y_q_dot = 0.0 ;//
const double Y_r_dot = -0.000398 ;// sign change??

const double Y_uu = 0.0 ;//
const double Y_uv = -0.027834 ;//
const double Y_uw = 0.0 ;//
const double Y_up = 0.0 ;//
const double Y_uq = 0.0 ;//

```

```

const double Y_ur = 0.005251 ;//

const double Y_uu_delta_rb = 0.0 ;// no bow rudder
const double Y_uu_delta_rs = 1.18E-2/2.0 ;//

const double Y_pq = 0.0 ;// (these aren't in Bahrke thesis model)
const double Y_qr = 0.0 ;//
const double Y_vq = 0.0 ;//
const double Y_wp = 0.0 ;//
const double Y_wr = 0.0 ;//
const double Y_vw = 0.0 ;//

const double C_dy = 0.5 ;// ??

//-----//
//      Heave equation of motion coefficients      //
//

const double Z_u_dot = 0.0 ;// Linear force coefficients acting in
const double Z_v_dot = 0.0 ;// the vertical body axis
const double Z_w_dot = -0.014529 ;// with respect to subscripted
const double Z_p_dot = 0.0 ;// motion components
const double Z_q_dot = -0.000633 ;//
const double Z_r_dot = 0.0 ;//

const double Z_vv = 0.0 ;//
const double Z_uw = -0.013910 ;//
const double Z_up = 0.0 ;//
const double Z_uq = -0.007545 ;//
const double Z_rr = 0.0 ;//
const double Z_pp = 0.0 ;//

const double Z_uu_delta_b = 0.0 ;//
const double Z_uu_delta_s = -0.005603 ;//

const double Z_pr = 0.0 ;// (these aren't in Bahrke thesis model)
const double Z_vp = 0.0 ;//

```

```

const double Z_vr = 0.0 ;//

const double Z_qn = 0.0 ;// no longer used in new model
const double Z_wn = 0.0 ;// no longer used in new model
const double Z_dsn = 0.0 ;// no longer used in new model

const double C_dz = 0.6 ;// ??

//-----//
//      Roll equation of motion coefficients      //

const double K_u_dot = 0.0 ;// Angular force coefficient acting
const double K_v_dot = 0.0 ;// about the longitudinal body axis
const double K_w_dot = 0.0 ;// with respect to subscripted
const double K_p_dot = -2.4E-4 ;// motion components
// NPS AUV value used since no SUBOFF value provided

const double K_q_dot = 0.0 ;//
const double K_r_dot = 0.0 ;//

const double K_uu = 0.0 ;//
const double K_uv = -0.000584 ;// NPS AUV is zero!
const double K_uw = 0.0 ;//
const double K_up = -5.4E-3 ;// surge-related roll damping drag
// NPS AUV value used since no SUBOFF value provided

const double K_uq = 0.0 ;//
const double K_ur = 0.0 ;//

const double K_uu_planes = 0.0 ;// (these aren't in Bahrke thesis model)
const double K_pq = 0.0 ;//
const double K_qr = 0.0 ;//
const double K_vq = 0.0 ;//
const double K_wp = 0.0 ;//
const double K_wr = 0.0 ;//
const double K_vw = 0.0 ;//

```



```
const double K_prop = 0.0 ; // K_prop "constant" no longer applicable
```

```
const double K_pn = 0.0 ; // no longer used in new model
```

```
const double K_pp = -2.02E-2 ; // test value for p-squared damping
                        // static roll damping drag
                        // NPS AUV value used since no SUBOFF value provided
```

```
const double K_p = K_pp/57.3 ; // estimate based on quadratic term
                        // (K_pp) equivalent damping at 1 deg/sec
```

```
//-----//
//      Pitch equation of motion coefficients      //
```

```
const double M_u_dot = 0.0 ; // Angular force coefficient acting
const double M_v_dot = 0.0 ; // about the athwartships body axis
const double M_w_dot = -0.000561 ; // with respect to subscripted
const double M_p_dot = 0.0 ; // motion components
const double M_q_dot = -0.000860 ; //
const double M_r_dot = 0.0 ; //
```

```
const double M_uu = 0.0 ; //
const double M_vv = 0.0 ; //
const double M_uw = -0.010324 ; //
const double M_pp = 0.0 ; //
const double M_rr = 0.0 ; //
```

```
const double M_uq = -0.003702 ; // surge-related pitch damping drag ***
```

```
const double M_uu_delta_bow = 0.0; // no bow rudder
```

```
const double M_uu_delta_stern = - x_rs * Z_uu_delta_s / 2.0;
                        //      note (-) Z_uu_delta_s
                        //      = - 0.058085219
```

```

const double M_pr = 0.0 ;// (these aren't in Bahrke thesis model)
const double M_vp = 0.0 ;//
const double M_vr = 0.0 ;//
const double M_prop = 0.0 ;// M_prop "constant" no longer applicable

const double M_qn = 0.0 ;// no longer used in new model
const double M_wn = 0.0 ;// no longer used in new model
const double M_dsn = 0.0 ;// no longer used in new model

const double M_qq = -7.00E-3 ;// slightly larger than N_rr estimate
// test value for q-squared
// static pitch damping drag
// estimated M_qq ~ K_pp * length / width
// Torsiello ~ 0.005* 7.3' / 10.1" = .005

const double M_q = M_qq / 57.3; // estimate based on quadratic term
// (M_qq) equivalent damping at 1 deg/sec

//-----//
// Yaw equation of motion coefficients //

const double N_u_dot = 0.0 ;// Angular force coefficient acting
const double N_v_dot = -0.000396 ;// about the vertical body axis
const double N_w_dot = 0.0 ;// with respect to subscripted
const double N_p_dot = 0.0 ;// motion components
const double N_q_dot = 0.0 ;//
const double N_r_dot = -0.000897 ;//

const double N_uu = 0.0 ;//
const double N_uv = -0.013648 ;// NPS AUV is zero!
const double N_uw = 0.0 ;//
const double N_up = 0.0 ;//
const double N_uq = 0.0 ;//
const double N_ur = -0.004444 ;// surge-related yaw damping drag

```

```
// N_uu_delta_rb and N_uu_delta_rs not symmetric due to different moment arms
```

```
const double N_uu_delta_rb = 0.0 ; // No bow rudder
```

```
const double N_uu_delta_rs = - x_rs * Y_uu_delta_rs / 2.0; //
```

```
const double N_prop = 0.0 ; // Normally 0.0 yaw moment due to paired  
// counter-rotating propellers;
```

```
// *** however N_prop is not zero if propellor rpms are independent
```

```
// thus yaw equation of motion now has yaw moments due to propellers
```

```
// and N_prop "constant" is no longer applicable
```

```
// *** value needed to account for single propeller twisting force
```

```
const double N_pq = 0.0 ; // (these aren't in Bahrke thesis model)
```

```
const double N_qr = 0.0 ; //
```

```
const double N_vq = 0.0 ; //
```

```
const double N_wp = 0.0 ; //
```

```
const double N_wr = 0.0 ; //
```

```
const double N_vw = 0.0 ; //
```

```
const double N_rr = -5.48E-3 ; // Torsiello value p.113 adjusted for L^5
```

```
// correction; static yaw damping drag
```

```
// estimated N_rr ~ M_qq * height/ width
```

```
// = .040 * 10.1" / 16.5"
```

```
// = 0.005
```

```
// Torsiello: 0.005
```

```
// Healey: N_rr ~ M_qq
```

```
// alternate N_rr = 2 * 2# * 1.92' / (rho/2 L^5 * r_max * r_max) => 0.0048473244
```

```
// using r_max = 16 deg/sec Torsiello which is consistent
```

```
const double N_r = N_rr / 57.3; // estimate based on quadratic term
```

```
// (N_rr) equivalent damping at 1 deg/sec
```

```
//-----//
```

```

// DEFINE THE LENGTH X, BREADTH bb, AND HEIGHT hh TERMS

const int cross_sections = 25;

// we must convert station coordinates (Roddy p. 16) to body coordinates
// center STATION value about zero then rescale to match actual size:
// (all units FEET)

#define STATION_TO_BODY(x) ((x - (20.4167/2.0)) * 14.2917 / 20.4167)

// Now a conversion function to convert B/B_x to feet (max diameter 1.667 ft)
// Note these values will be the same for hh and bb since SUBOFF is a cylinder

#define B_BX_TO_FEET(y) (y * 1.16667)

double xx [cross_sections] = {
    STATION_TO_BODY(0.0),
    STATION_TO_BODY(0.1),
    STATION_TO_BODY(0.2),
    STATION_TO_BODY(0.3),
    STATION_TO_BODY(0.4),
    STATION_TO_BODY(0.5),
    STATION_TO_BODY(0.6),
    STATION_TO_BODY(0.7),
    STATION_TO_BODY(1.0),
    STATION_TO_BODY(2.0),
    STATION_TO_BODY(3.0),
    STATION_TO_BODY(4.0),
    STATION_TO_BODY(7.7143),
    STATION_TO_BODY(10.0),
    STATION_TO_BODY(15.1429),
    STATION_TO_BODY(16.0),
    STATION_TO_BODY(17.0),
    STATION_TO_BODY(18.0),
    STATION_TO_BODY(19.0),
    STATION_TO_BODY(20.0),

```

```

STATION_TO_BODY(20.1),
STATION_TO_BODY(20.2),
STATION_TO_BODY(20.3),
STATION_TO_BODY(20.4),
STATION_TO_BODY(20.4167)
};

```

```

double hh [cross_sections] = {
B_BX_TO_FEET(0.00000),
B_BX_TO_FEET(0.29058),
B_BX_TO_FEET(0.39396),
B_BX_TO_FEET(0.46600),
B_BX_TO_FEET(0.52147),
B_BX_TO_FEET(0.56627),
B_BX_TO_FEET(0.60352),
B_BX_TO_FEET(0.63514),
B_BX_TO_FEET(0.70744),
B_BX_TO_FEET(0.84713),
B_BX_TO_FEET(0.94066),
B_BX_TO_FEET(0.99282),
B_BX_TO_FEET(1.00000),
B_BX_TO_FEET(1.00000),
B_BX_TO_FEET(1.00000),
B_BX_TO_FEET(0.97598),
B_BX_TO_FEET(0.81910),
B_BX_TO_FEET(0.55025),
B_BX_TO_FEET(0.26835),
B_BX_TO_FEET(0.11724),
B_BX_TO_FEET(0.11243),
B_BX_TO_FEET(0.10074),
B_BX_TO_FEET(0.07920),
B_BX_TO_FEET(0.03178),
B_BX_TO_FEET(0.00000)
};

```

```

// note identical since cylindrical

```

```

double bb [cross_sections] = {
  B_BX_TO_FEET(0.00000),
  B_BX_TO_FEET(0.29058),
  B_BX_TO_FEET(0.39396),
  B_BX_TO_FEET(0.46600),
  B_BX_TO_FEET(0.52147),
  B_BX_TO_FEET(0.56627),
  B_BX_TO_FEET(0.60352),
  B_BX_TO_FEET(0.63514),
  B_BX_TO_FEET(0.70744),
  B_BX_TO_FEET(0.84713),
  B_BX_TO_FEET(0.94066),
  B_BX_TO_FEET(0.99282),
  B_BX_TO_FEET(1.00000),
  B_BX_TO_FEET(1.00000),
  B_BX_TO_FEET(1.00000),
  B_BX_TO_FEET(0.97598),
  B_BX_TO_FEET(0.81910),
  B_BX_TO_FEET(0.55025),
  B_BX_TO_FEET(0.26835),
  B_BX_TO_FEET(0.11724),
  B_BX_TO_FEET(0.11243),
  B_BX_TO_FEET(0.10074),
  B_BX_TO_FEET(0.07920),
  B_BX_TO_FEET(0.03178),
  B_BX_TO_FEET(0.00000)
};

#undef STATION_TO_BODY(x)
#undef B_BX_TO_FEET(y)

//-----//

#endif // UUVMODEL_H

```

## APPENDIX B - GENERIC SUBMARINE MODEL COEFFICIENTS

//

/\*

Program: SUBOFFmodel.H (Version of UUVmodel.H for DARPA SUBOFF model)

Author: Don Brutzman

Revised: 16 August 95

System: Irix 5.3

Compiler: ANSI C++

Compilation: irix> cp SUBOFFmodel.H UUVmodel.H

irix> make dynamics

Advisors: Dr. Mike Zyda, Dr. Bob McGhee and Dr. Tony Healey

References: Healey, Anthony J. and Lienard, David, "Multivariable  
Sliding Mode Control for Autonomous Diving and Steering  
of Unmanned Underwater Vehicles," IEEE Journal of Oceanic  
Engineering, vol. 18 no. 3, July 1993, pp. 327-339,

Lewis, Edward V., editor, \_Principles of Naval  
Architecture volume III\_, second revision, The Society of  
Naval Architects and Marine Engineers, Jersey City  
New Jersey, 1988, pp. 188-190 and 418-423.

Gertler, Morton and Hagen, Grant R., \_Standard Equations  
of Motion for Submarine Simulation\_, Naval Ship  
Research and Development Center (NSRDC) Research and  
Development Report 2510, Washington DC, June 1967.

Smith, N.S., Crane J.W. and Summey, D.C., \_SDV Simulator Hydrodynamic Coefficients\_, Naval Coastal Systems Center (NCSC), Panama City Florida, June 1978. Declassified.

Marco, David. "Slow Speed Control and Dynamic Positioning of an Autonomous Vehicle," Ph.D. dissertation, Naval Postgraduate School, Monterey California, March 1995.

Bahrke, Fredric G., "On-Line Identification of the Speed, Steering and Diving Response Parameters of an Autonomous Underwater Vehicle from Experimental Data," Master's Thesis, Naval Postgraduate School, Monterey California, March 1992.

Warner, David C., "Design, Simulation and Experimental Verification of a Computer Model and Enhanced Position Estimator for the NPS AUV II," Master's Thesis, Naval Postgraduate School, Monterey California, December 1991.

Bacon, Daniel K. Jr., "Integration of a Submarine into NPSNET," Master's Thesis, Naval Postgraduate School, Monterey California, September 1995.

Roddy, Robert F., "Investigation of the Stability and Control Characteristics of Several Configurations of the DARPA SUBOFF Model (DTRC Model 5470) from Captive-Model Experiments," Technical Report DTRC/SHD-1298-08, Ship Hydrodynamics Department, David Taylor Research Center (DTRC), September 1990 (unclassified).

Model note: We use values for DARPA SUBOFF model in Configuration 2 (fully appended) which includes plane surfaces.

Notes: const definitions are for software engineering reliability they can be changed to variables if coefficient modification becomes desirable





```

const double I_y   = 0.001053*m4_ft4 ; // Nms^2   = I_yy =
const double I_z   = 0.001084*m4_ft4 ; // Nms^2   = I_zz =
const double I_xy  = 0.0   ; // Nms^2   = I_yx
const double I_xz  = 0.0   ; // Nms^2   = I_zx
const double I_yz  = 0.0   ; // Nms^2   = I_zy

#undef      m4_ft4

//          Centers of Gravity & Buoyancy
const double x_G   = 0.556   *0.3048; // m
const double y_G   = 0.0     *0.3048; // m
const double z_G   = 0.0     *0.3048; // m    Note CG below CB  Marco 0.5"
const double x_B   = 0.532094*0.3048; // m
const double y_B   = 0.0     *0.3048; // m
const double z_B   = -0.006669*0.3048; // m    CB at center of UUV

//          Additional hull characteristics          //

const double H     = 0.240792 ; // m    main hull height 9.50"

double revisedBuoyancy, revised_x_B;

double surface_length = 0.0; // distances (CB to surface) & (CB to nose)
// along body axis
const double nose_length = (0.90 * L) / 2.0;

#else // (not SI) standard British units -----

const double Weight = 6900.0 * 2000.0 ; // lb    Weight
const double Buoyancy = 6900.0 * 2000.0 ; // lb    Buoyancy
const double L      = 360.0      ; // ft    characteristic length

const double g      = 32.174 ; // ft/s^2    gravitational constant
const double rho    = 1.94   ; // slugs/ft^3 mass density of fresh water
const double m      = Weight / g ; // lb/ft-s^2 vehicle mass incl. free flood
// =

```

```

// Moments of inertia units normalized using perpendicular length 13.9792 ft
// (Roddy p.3, Feldman p. 6)

#define Lnorm(i) (i * 0.5*rho*360.0*360.0*360.0*360.0*360.0)

// naive approach to scaling up moments of inertia

                                ; //      Inertia matrix coefficients

// new value I_x not found, verify through calculations ??
const double I_x    = Lnorm(0.000060); // lb-ft-sec^2    =I_xx

const double I_y    = Lnorm(0.001053); // lb-ft-sec^2    =I_yy
const double I_z    = Lnorm(0.001084); // lb-ft-sec^2    =I_zz
const double I_xy   = Lnorm(0.0)      ; // lb-ft-sec^2    =I_yx
const double I_xz   = Lnorm(0.0)      ; // lb-ft-sec^2    =I_zx
const double I_yz   = Lnorm(0.0)      ; // lb-ft-sec^2    =I_zy

#undef Lnorm(i)

                                //      Centers of Gravity & Buoyancy
const double x_G    = 0.0      ; // ft      listed as 0.556
const double y_G    = 0.0      ; // ft

// *** modification - z_G moved down from 0.0 for more adequate righting arm ***
const double z_G    = 3.0      ; // ft      Note CG below CB

const double x_B    = 0.0      ; // ft      0.010416667
const double y_B    = 0.0      ; // ft
const double z_B    = 0.0      ; // ft

// Thruster/propeller distances from centerlines. Note stern/port are negative.

const double x_bow_vertical = 0.0 ; // ft No thrusters!
const double x_stern_vertical = 0.0 ; // ft

```

```

const double x_bow_lateral = 0.0 ; // ft
const double x_stern_lateral = 0.0 ; // ft

const double y_port_propeller = 0.0 ; // ft   Single propeller, on centerline
const double y_stbd_propeller = 0.0 ; // ft

// Rudder bow/stern distances from centerlines. 0.5 is all the way forward/aft.

const double x_rb = 0.0 * L; // proportional distance to bow (none!)
const double x_rs = -0.427 * L; // proportional distance to stern

//      Additional hull characteristics      //

const double H = 32.0 ; // ft      main hull diameter 32.0 ft

double revisedBuoyancy, revised_x_B;

double surface_length = 0.0; // distances (CB to surface) & (CB to nose)
// along body axis
const double nose_length = (0.90 * L) / 2.0;

const int THRUSTERS = FALSE; // are cross-body thrusters present?

#endif

//-----//
//      Surge equation of motion coefficients      //

// *** warning: no X_ coefficients found in Roddy reference
const double X_u_dot = 0.0 ; // Linear force coefficients acting in
const double X_v_dot = 0.0 ; // the longitudinal body axis
const double X_w_dot = 0.0 ; // with respect to subscripted
const double X_p_dot = 0.0 ; // motion components
const double X_q_dot = 0.0 ; //
const double X_r_dot = 0.0 ; //

```

```

const double X_uu = 0.0 ;//
const double X_vv = 0.0 ;//
const double X_ww = 0.0 ;//
const double X_pp = 0.0 ;//
const double X_qq = 0.0 ;//
const double X_rr = 0.0 ;//

const double X_prop = 0.0 ;// X_prop "constant" no longer applicable

// plane surface drags not given, either 0 or estimated at 1/2 AUV effectiveness
// (same swag factor for other coefficients later)

const double X_uu_delta_b_delta_b = 0.0 ;// drag due to bow plane
const double X_uu_delta_s_delta_s = -1.018E-2/2.0 ;// drag due to stern plane
const double X_uu_delta_r_delta_r = -1.018E-2/2.0 ;// drag due to rudder

const double X_pr = 0.0 ;// (these aren't in Bahrke thesis model)
const double X_wq = 0.0 ;//
const double X_vp = 0.0 ;//
const double X_vr = 0.0 ;//

const double X_uq_delta_bow = 0.0 ;//
const double X_uq_delta_stern = 0.0 ;//
const double X_ur_delta_rudder = 0.0 ;//
const double X_uv_delta_rudder = 0.0 ;//
const double X_uw_delta_bow = 0.0 ;//
const double X_uw_delta_stern = 0.0 ;//

const double X_qdsn = 0.0 ;// no longer used in new model
const double X_wdsn = 0.0 ;// no longer used in new model
const double X_dsdsn = 0.0 ;// no longer used in new model

// we assume 20 knot max speed = 2000 yds/3 min = 2000 ft/min = 33.333 ft/sec
// we assume max rpm is 200

const double speed_per_rpm = 33.333 / 200.0 ;// steady state: 0.16667

```

// = (33.3 feet/sec) per 200 rpm

```
const double MAX_RPM      = 200.0 ;//
```

```
// *** recheck this value:
```

```
const double C_d0  = 0.00778      ;//
```

```
//-----//
```

```
//      Sway equation of motion coefficients      //
```

```
const double Y_u_dot = 0.0      ;// Linear force coefficients acting in
```

```
const double Y_v_dot = -0.016186 ;// the athwartships body axis
```

```
const double Y_w_dot = 0.0      ;// with respect to subscripted
```

```
const double Y_p_dot = 0.0      ;// motion components
```

```
const double Y_q_dot = 0.0      ;//
```

```
const double Y_r_dot = -0.000398 ;// sign change??
```

```
const double Y_uu  = 0.0      ;//
```

```
const double Y_uv  = -0.027834 ;//
```

```
const double Y_uw  = 0.0      ;//
```

```
const double Y_up  = 0.0      ;//
```

```
const double Y_uq  = 0.0      ;//
```

```
const double Y_ur  = 0.005251 ;//
```

```
const double Y_uu_delta_rb = 0.0 ;// no bow rudder
```

```
const double Y_uu_delta_rs = 1.18E-2/2.0 ;//
```

```
const double Y_pq  = 0.0      ;// (these aren't in Bahrke thesis model)
```

```
const double Y_qr  = 0.0      ;//
```

```
const double Y_vq  = 0.0      ;//
```

```
const double Y_wp  = 0.0      ;//
```

```
const double Y_wr  = 0.0      ;//
```

```
const double Y_vw  = 0.0      ;//
```

```
const double C_dy  = 0.5      ;// ??
```

```

//-----//
//      Heave equation of motion coefficients      //

const double Z_u_dot = 0.0 ;// Linear force coefficients acting in
const double Z_v_dot = 0.0 ;// the vertical body axis
const double Z_w_dot = -0.014529 ;// with respect to subscripted
const double Z_p_dot = 0.0 ;// motion components
const double Z_q_dot = -0.000633 ;//
const double Z_r_dot = 0.0 ;//

const double Z_vv = 0.0 ;//
const double Z_uw = -0.013910 ;//
const double Z_up = 0.0 ;//
const double Z_uq = -0.007545 ;//
const double Z_rr = 0.0 ;//
const double Z_pp = 0.0 ;//

const double Z_uu_delta_b = 0.0 ;//
const double Z_uu_delta_s = -0.005603 ;//

const double Z_pr = 0.0 ;// (these aren't in Bahrke thesis model)
const double Z_vp = 0.0 ;//
const double Z_vr = 0.0 ;//

const double Z_qn = 0.0 ;// no longer used in new model
const double Z_wn = 0.0 ;// no longer used in new model
const double Z_dsn = 0.0 ;// no longer used in new model

const double C_dz = 0.6 ;// ??

//-----//
//      Roll equation of motion coefficients      //

const double K_u_dot = 0.0 ;// Angular force coefficient acting
const double K_v_dot = 0.0 ;// about the longitudinal body axis
const double K_w_dot = 0.0 ;// with respect to subscripted

```

```

const double K_p_dot = -2.4E-4 ;//      motion components
      // NPS AUV value used

const double K_q_dot = 0.0 ;//
const double K_r_dot = 0.0 ;//

const double K_uu = 0.0 ;//
const double K_uv = -0.000584 ;//
const double K_uw = 0.0 ;//
const double K_up = -5.4E-3 ;// surge-related roll damping drag
      // NPS AUV value used

const double K_uq = 0.0 ;//
const double K_ur = 0.0 ;//

const double K_uu_planes = 0.0 ;// (these aren't in Bahrke thesis model)
const double K_pq = 0.0 ;//
const double K_qr = 0.0 ;//
const double K_vq = 0.0 ;//
const double K_wp = 0.0 ;//
const double K_wr = 0.0 ;//
const double K_vw = 0.0 ;//
const double K_prop = 0.0 ;// K_prop "constant" no longer applicable

const double K_pn = 0.0 ;// no longer used in new model

const double K_pp = -2.02E-2 ;// test value for p-squared damping
      // static roll damping drag
      // NPS AUV value used

const double K_p = K_pp/57.3 ;// estimate based on quadratic term
      // (K_pp) equivalent damping at 1 deg/sec

//-----//
//      Pitch equation of motion coefficients      //

```



```

const double M_u_dot = 0.0 ;// Angular force coefficient acting
const double M_v_dot = 0.0 ;// about the athwartships body axis
const double M_w_dot = -0.000561 ;// with respect to subscripted
const double M_p_dot = 0.0 ;// motion components
const double M_q_dot = -0.000860 ;//
const double M_r_dot = 0.0 ;//

const double M_uu = 0.0 ;//
const double M_vv = 0.0 ;//
const double M_uw = -0.010324 ;//
const double M_pp = 0.0 ;//
const double M_rr = 0.0 ;//

const double M_uq = -0.003702 ;// surge-related pitch damping drag ***

const double M_uu_delta_bow = 0.0;

const double M_uu_delta_stern = - x_rs * Z_uu_delta_s / 2.0;
// note (-) Z_uu_delta_s
// = - 0.058085219

const double M_pr = 0.0 ;// (these aren't in Bahrke thesis model)
const double M_vp = 0.0 ;//
const double M_vr = 0.0 ;//
const double M_prop = 0.0 ;// M_prop "constant" no longer applicable

const double M_qn = 0.0 ;// no longer used in new model
const double M_wn = 0.0 ;// no longer used in new model
const double M_dsn = 0.0 ;// no longer used in new model

const double M_qq = -7.00E-3 ;// slightly larger than N_rr estimate
// test value for q-squared
// static pitch damping drag
// estimated M_qq ~ K_pp * length / width
// Torsiello ~ 0.005* 7.3' / 10.1" = .005

```

```

const double M_q   = M_qq / 57.3; // estimate based on quadratic term
                        // (M_qq) equivalent damping at 1 deg/sec

//-----//
//      Yaw  equation of motion coefficients      //

const double N_u_dot =  0.0   ;// Angular force coefficient acting
const double N_v_dot = -0.000396 ;// about the vertical body axis
const double N_w_dot =  0.0   ;// with respect to subscripted
const double N_p_dot =  0.0   ;// motion components
const double N_q_dot =  0.0   ;//
const double N_r_dot = -0.000897 ;//

const double N_uu   =  0.0   ;//
const double N_uv   = -0.013648 ;//
const double N_uw   =  0.0   ;//
const double N_up   =  0.0   ;//
const double N_uq   =  0.0   ;//
const double N_ur   = -0.004444 ;// surge-related yaw damping drag

// N_uu_delta_rb and N_uu_delta_rs not symmetric due to different moment arms

const double N_uu_delta_rb = 0.0 ;// No bow rudder
const double N_uu_delta_rs = - x_rs * Y_uu_delta_rs / 2.0; //

const double N_prop =  0.0   ;// Normally 0.0 yaw moment due to paired
                        // counter-rotating propellers;
                        // however N_prop is not zero if propellor rpms are independent
                        // thus yaw equation of motion now has yaw moments due to propellers
                        // and N_prop "constant" is no longer applicable

const double N_pq   =  0.0   ;// (these aren't in Bahrke thesis model)
const double N_qr   =  0.0   ;//
const double N_vq   =  0.0   ;//

```

```

const double N_wp = 0.0 ;//
const double N_wr = 0.0 ;//
const double N_vw = 0.0 ;//

const double N_rr = -5.48E-3 ;// Torsiello value p.113 adjusted for L^5
// correction; static yaw damping drag
// estimated N_rr ~ M_qq * height/ width
//          = .040 * 10.1" / 16.5"
//          = 0.005
// Torsiello: 0.005
// Healey: N_rr ~ M_qq

// alternate N_rr = 2 * 2# * 1.92' / (rho/2 L^5 * r_max * r_max) => 0.0048473244
//          using r_max = 16 deg/sec Torsiello which is consistent

const double N_r = N_rr / 57.3; // estimate based on quadratic term
// (N_rr) equivalent damping at 1 deg/sec

//-----//

// DEFINE THE LENGTH X, BREADTH bb, AND HEIGHT hh TERMS

const int cross_sections = 25;

// we must convert station coordinates (Roddy p. 16) to body coordinates
// center STATION value about zero then rescale to match actual size:

#define STATION_TO_BODY(x) ((x - (20.4167/2.0)) * 14.2917 / 20.4167)

// Now a conversion function to convert B/B_x to feet (max diameter 1.667 ft)
// Note these values will be the same for hh and bb since SUBOFF is a cylinder

#define B_BX_TO_FEET(y) (y * 1.16667)

double xx [cross_sections] = {
    STATION_TO_BODY(0.0),

```

```

STATION_TO_BODY(0.1),
STATION_TO_BODY(0.2),
STATION_TO_BODY(0.3),
STATION_TO_BODY(0.4),
STATION_TO_BODY(0.5),
STATION_TO_BODY(0.6),
STATION_TO_BODY(0.7),
STATION_TO_BODY(1.0),
STATION_TO_BODY(2.0),
STATION_TO_BODY(3.0),
STATION_TO_BODY(4.0),
STATION_TO_BODY(7.7143),
STATION_TO_BODY(10.0),
STATION_TO_BODY(15.1429),
STATION_TO_BODY(16.0),
STATION_TO_BODY(17.0),
STATION_TO_BODY(18.0),
STATION_TO_BODY(19.0),
STATION_TO_BODY(20.0),
STATION_TO_BODY(20.1),
STATION_TO_BODY(20.2),
STATION_TO_BODY(20.3),
STATION_TO_BODY(20.4),
STATION_TO_BODY(20.4167)
};

```

```

double hh [cross_sections] = {
B_BX_TO_FEET(0.00000),
B_BX_TO_FEET(0.29058),
B_BX_TO_FEET(0.39396),
B_BX_TO_FEET(0.46600),
B_BX_TO_FEET(0.52147),
B_BX_TO_FEET(0.56627),
B_BX_TO_FEET(0.60352),
B_BX_TO_FEET(0.63514),
B_BX_TO_FEET(0.70744),

```

```

B_BX_TO_FEET(0.84713),
B_BX_TO_FEET(0.94066),
B_BX_TO_FEET(0.99282),
B_BX_TO_FEET(1.00000),
B_BX_TO_FEET(1.00000),
B_BX_TO_FEET(1.00000),
B_BX_TO_FEET(0.97598),
B_BX_TO_FEET(0.81910),
B_BX_TO_FEET(0.55025),
B_BX_TO_FEET(0.26835),
B_BX_TO_FEET(0.11724),
B_BX_TO_FEET(0.11243),
B_BX_TO_FEET(0.10074),
B_BX_TO_FEET(0.07920),
B_BX_TO_FEET(0.03178),
B_BX_TO_FEET(0.00000)
};

```

```

double bb [cross_sections] = {
B_BX_TO_FEET(0.00000),
B_BX_TO_FEET(0.29058),
B_BX_TO_FEET(0.39396),
B_BX_TO_FEET(0.46600),
B_BX_TO_FEET(0.52147),
B_BX_TO_FEET(0.56627),
B_BX_TO_FEET(0.60352),
B_BX_TO_FEET(0.63514),
B_BX_TO_FEET(0.70744),
B_BX_TO_FEET(0.84713),
B_BX_TO_FEET(0.94066),
B_BX_TO_FEET(0.99282),
B_BX_TO_FEET(1.00000),
B_BX_TO_FEET(1.00000),
B_BX_TO_FEET(1.00000),
B_BX_TO_FEET(0.97598),
B_BX_TO_FEET(0.81910),

```

```
B_BX_TO_FEET(0.55025),  
B_BX_TO_FEET(0.26835),  
B_BX_TO_FEET(0.11724),  
B_BX_TO_FEET(0.11243),  
B_BX_TO_FEET(0.10074),  
B_BX_TO_FEET(0.07920),  
B_BX_TO_FEET(0.03178),  
B_BX_TO_FEET(0.00000)  
};  
  
#undef STATION_TO_BODY(x)  
#undef B_BX_TO_FEET(y)
```

```
//-----//
```

```
#endif // UUVMODEL_H
```

## **APPENDIX C - OBTAINING NPSNET SOURCE CODE**

### **The NPSNET Email Addresses**

For general code questions, concerns, comments, requests for distributions and documentation, and bug reports, email [npsnet@cs.nps.navy.mil](mailto:npsnet@cs.nps.navy.mil).

To contact principal investigators, receive overall research project information and funding, or request demonstrations, email [npsnet-info@cs.nps.navy.mil](mailto:npsnet-info@cs.nps.navy.mil).

### **Anonymous FTP Archives**

A number of sites maintain archives of documents and software that are generally available. The procedure is to FTP to these sites, using the name "anonymous" with a password of "guest" or your email address, as prompted. The following sites are known to us to have information relevant to NPSNET, computer graphics, or VR.

#### **Site Host Directory Notes**

NPS [cs.nps.navy.mil pub/barham](ftp://cs.nps.navy.mil/pub/barham) Retrieve README for instructions on obtaining NPSNET.

SGI [sgigate.sgi.com pub/Performer](ftp://sgigate.sgi.com/pub/Performer) Performer stuff.

ISI [ftp.isi.edu mbone/faq.txt](ftp://ftp.isi.edu/mbone/faq.txt) is the MBONE FAQ.

IST [tiig.ist.ucf.edu public/DIS](ftp://tiig.ist.ucf.edu/public/DIS) stuff.

XEROX-PARC [parcftp.xerox.com pub/net-research](ftp://parcftp.xerox.com/pub/net-research) Multicast software for the MBONE; MBONE maps.

### **World-Wide Web**

The WWW is a collection of sites that make information available through use of browsers like Mosaic or Netscape. Following are the sites we know of that have information relevant to NPSNET, computer graphics, or VR.

The NPSNET Research Group home page

URL: [ftp://cs.nps.navy.mil/pub/NPSNET\\_MOSAIC/npsnet\\_mosaic.html](ftp://cs.nps.navy.mil/pub/NPSNET_MOSAIC/npsnet_mosaic.html)

Notes: Relevant NPSNET documents, including papers and theses produced at NPS, which are available for downloading.

#### NPSNET Distribution Information

URL: <http://cs.nps.navy.mil/research/npsnet/distribution/page.html>

Notes: Info on NPSNET distribution and documentation



## LIST OF REFERENCES

Barham, Paul T., Zyda, Michael J., Pratt, David R., Locke, John, Falby, John, "*NPSNET-IV: A DIS-Compatible, Object-Oriented Software Architecture for Virtual Environments*", unpublished paper, Naval Postgraduate School, Monterey, California October 1994.

Brutzman, Donald P., *A Virtual World for an Autonomous Underwater Vehicle*, Dissertation, Naval Postgraduate School, Monterey, California March 1994. Available at <http://www.stl.nps.navy.mil/~brutzman/dissertation>

Covington, James H., "*Implementing an Open Ocean Theater in NPSNET*", Master's Thesis, Naval Postgraduate School, Monterey, California March 1994. Available at <http://www.nps.navy.mil/res...ications/covington.thesis.ps.Z>

Dalton, John H., Secretary of the Navy, "*Forward...From the Sea*", Doctrine Policy Paper, September 1994.

Healey, Antony J., "Dynamics of Marine Vehicles", Naval Postgraduate School, Monterey, California unpublished course notes 1993.

Hearn, John H., "*NPSNET: Physically Based, Autonomous, Naval Surface Agents*", Master's Thesis, Naval Postgraduate School, Monterey, California September 1993.

Jurewicz, T., "*A Real Time Autonomous Underwater Vehicle Dynamic Simulator*", Master's Thesis, Naval Postgraduate School, Monterey, California June 1989.

McMahan, Christopher B., "*NPSNET-IV: An Object-Oriented Interface for a Three-Dimensional Virtual World*", Master's Thesis, Naval Postgraduate School, Monterey, California December 1994.

Nobles, Joesph and Garrova, James, "*Virtual Shipboard Navigational Trainer*" Master's Thesis, Naval Postgraduate School, Monterey, California June 1995.

Roddy, Robert F., "*Investigation of the Stability and Control Characteristics of Several Configurations of the DARPA SUBOFF Model (DTRC MODEL 5470) from Captive-Model Experiments*", Ship Hydromechanics Departmental Report DTRC/SHD-1298-08, David Taylor Research Center, Bethesda< Maryland Septmeber 1990.

Sharpe, Richard, *Jane's Fighting Ships*, Jane's Information Group Ltd., Alexandria, Virginia 1993.

Schmidt, Dennis A., "*NPSNET: A Graphical Based Expert System To Model P-3 Aircraft Interaction With Submarines and Ships*", Master's Thesis, Naval Postgraduate School, Monterey, California June 1993.

Schneiderman, Ben, *Designing The User Interface: Strategies For Effective Human-Computer Interaction*, Addison-Wesley, Reading, Massachusetts 1992.

Sullivan, J., Frost, D., *U.S. Military Simulation and Training Markets*, Commercial Survey, Frost & Sullivan, Mountain View, California January 1993.

Young, Roy D., "*NPSNET: A Real-Time 3D Interactive Virtual World*", Master's Thesis, Naval Postgraduate School, Monterey, California September 1993.

Zehner, Stanley N., "*Modeling and Simulation Of A Deep Submergence Rescue Vehicle (DSRV) And Its Networked Application*", Master's Thesis, Naval Postgraduate School, Monterey, California June 1993.

Zeswitz, S. R., "*NPSNET: Integration of Distributed Interactive Simulation (DIS) Protocol for Communication Architecture and Information Interchange*", Master's Thesis, Naval Postgraduate School, Monterey, California September 1993. Available at <http://www.nps.navy.mil/res...ications/Steve.Zeswitz.thesis.ps.Z>

Zyda, Michael J., Jurewicz, Thomas A., Floyd, Charles A., and McGhee, Robert B., "Physically Based Modeling of Rigid Body Motion in a Real-Time Graphical Simulator", unpublished paper, Naval Postgraduate School, Monterey, California September 1991.

Zyda, Michael J., Pratt, David R., Kelleher, Kristen M., *1994 Annual Report for the NPSNET Research Group*, Naval Postgraduate School, Monterey, California 1994. Available at <http://www.nps.navy.mil/re...994.NPSNET.Annual.Report.ps.Z>

## INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center ..... 2  
Cameron Station  
Alexandria, VA 22304-6145
2. Dudley Knox Library ..... 2  
Code 52  
Naval Postgraduate School  
Monterey, CA 93943-5101
3. Dr Ted Lewis, Chairman, Code CS ..... 2  
Computer Science Department  
Naval Postgraduate School  
Monterey, CA 93943
4. Dr Michael Zyda, Code CS/ZY ..... 1  
Computer Science Department  
Naval Postgraduate School  
Monterey, CA 93943
5. Dr Donald Brutzman, Code UW/BR ..... 1  
Undersea Warfare Department  
Naval Postgraduate School  
Monterey, CA 93943
6. Mr John Falby, Code CS CS/FA ..... 1  
Computer Science Department  
Naval Postgraduate School  
Monterey, CA 93943
7. Lt Daniel K. Bacon. .... 1  
12430 Rue Cheaumont  
San Diego, CA 92131